

Aspect-Oriented User Requirements Notation

Gunter Mussbacher

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Ph.D. in Computer Science

Under the auspices of the Ottawa-Carleton Institute for Computer Science



University of Ottawa
Ottawa, Ontario, Canada
November 2010

© Gunter Mussbacher, Ottawa, Canada, 2010

Abstract

Technologies based on aspects and applied at the early stages of software development allow requirements engineers to better encapsulate crosscutting concerns in requirements models. The Aspect-oriented User Requirements Notation (AoURN) extends the User Requirements Notation (URN) with aspects and thus unifies goal-oriented, scenario-based, and aspect-oriented concepts in one graphical framework. As URN is a standard of the International Telecommunication Union, AoURN is the first standards-based and graphical approach that integrates the three aforementioned techniques. Minimal changes to URN ensure that requirements engineers can continue working with goal and scenario models expressed in a familiar notation. With AoURN, concerns in goal and scenario models, regardless of whether these concerns crosscut or not, can be managed across model types. Typical concerns in URN are non-functional requirements, use cases, and stakeholder goals. As AoURN expresses concern composition rules with URN itself, it is possible to describe rules in an exhaustive and highly flexible way that is not restricted by any specific composition language. In addition, AoURN employs a composition technique that is not based solely on language syntax but takes semantics into account, and therefore is more robust to refactoring of an aspect-oriented model. Since composition based on semantics requires precisely defined languages, the semantics of URN is clarified and the concrete syntax enhanced to capture identified semantic variations of the language. A qualitative comparison of aspect-oriented techniques for scenario-based and goal-oriented requirements engineering is presented. An evaluation carried out based on metrics adapted from literature and a task-based evaluation suggest that AoURN models are more scalable than URN models and exhibit better modularity, reusability, and maintainability. A proof-of-concept implementation of AoURN in the jUCMNav tool is also discussed. While AoURN's ability to encapsulate concerns across model types further bridges the gap between goals and scenarios, Early Aspects research benefits from a standardized way of modeling concerns with AoURN.

Acknowledgment

I wish to thank foremost my supervisor and great friend, Daniel Amyot, for a thoroughly enjoyable, motivating, and productive research experience over the last years. My committee members, Tim Lethbridge, Bran Selic, and Stéphane Somé, as well as my external examiner, Dan Berry, are thanked for accepting to review this work and for their useful and constructive comments and feedback.

This work was shaped through many discussions and collaborations I was fortunate to have with colleagues and co-authors – too numerous to be listed, but I am sure you know who you are! As representatives of all these colleagues and co-authors, I would like to express my gratitude to Ana Moreira and João Araújo in Lisbon as well as Jon Whittle in Lancaster for their intense collaborations and for welcoming me to their homes and making me comfortable during my research visits at their wonderful universities.

My proof-of-concept implementation of AoURN features in the jUCMNav tool went smoothly thanks to the highly-talented Jason Kealey and Etienne Tremblay. Jason and Etienne are to a great extent responsible for building the tool's excellent main code base which made it easy to be extended. Furthermore, their company LavaBlast Software Inc. also contributed to some of the AoURN features in the context of a major usability improvement project for jUCMNav.

A big thank you goes out to the always helpful system support and administration staff of SITE and in particular Jacques Sincennes for his immense trouble-shooting skills.

This work was made possible by the financial support of the Natural Sciences and Engineering Research Council of Canada, the Ontario Graduate Scholarship Program, and the University of Ottawa.

Finally, Kiran – thank you!

Table of Contents

Abstract	i
Acknowledgment	ii
Table of Contents	iii
List of Figures.....	viii
List of Tables	xii
List of Acronyms	xiii
Chapter 1. Introduction.....	1
1.1. <i>Motivation</i>	<i>1</i>
1.2. <i>Research Hypothesis.....</i>	<i>2</i>
1.3. <i>Thesis Contributions</i>	<i>3</i>
1.4. <i>Publications Based on Thesis.....</i>	<i>4</i>
1.5. <i>Formatting Conventions.....</i>	<i>7</i>
1.6. <i>Thesis Outline</i>	<i>8</i>
Chapter 2. Background.....	9
2.1. <i>User Requirements Notation</i>	<i>9</i>
2.1.1 <i>Goal-Oriented Requirement Language.....</i>	<i>11</i>
2.1.2 <i>Use Case Maps.....</i>	<i>14</i>
2.2. <i>Basic Concepts of Aspect-Oriented Software Development.....</i>	<i>19</i>
2.3. <i>Summary.....</i>	<i>22</i>
Chapter 3. Use Case Maps 2.0.....	23
3.1. <i>Motivating the Need for Use Case Maps 2.0.....</i>	<i>23</i>
3.2. <i>Workflow Patterns</i>	<i>25</i>
3.3. <i>Clarifying the Semantics of Use Case Maps</i>	<i>26</i>
3.4. <i>Synchronizing Stub.....</i>	<i>30</i>
3.5. <i>Instances of UCM Maps.....</i>	<i>33</i>
3.5.1 <i>Singleton Maps</i>	<i>33</i>
3.5.2 <i>Synchronizing Stubs and Instances of UCM Maps.....</i>	<i>36</i>

3.6.	<i>Plug-in Bindings for UCM Components and Responsibilities</i>	40
3.6.1	Component Plug-in Bindings.....	40
3.6.2	Responsibility Plug-in Bindings	43
3.7.	<i>Modeling Cancellations and Exceptions in Use Case Maps</i>	44
3.7.1	Explicit Failure Points and Abort Mechanisms.....	45
3.7.2	Implicit Failure Points	50
3.8.	<i>Metamodel of Use Case Maps 2.0</i>	51
3.9.	<i>Summary</i>	54
Chapter 4.	Assessment of Use Case Maps 2.0	55
4.1.	<i>Basic Control Flow Patterns</i>	55
4.2.	<i>Advanced Branching and Synchronization Patterns</i>	57
4.3.	<i>Multiple Instance Patterns</i>	61
4.4.	<i>State-Based Patterns</i>	63
4.5.	<i>Cancellation and Force Completion Patterns</i>	65
4.6.	<i>Iteration Patterns</i>	67
4.7.	<i>Termination Patterns</i>	68
4.8.	<i>Trigger Patterns</i>	69
4.9.	<i>Assessment Results</i>	69
4.10.	<i>Summary</i>	73
Chapter 5.	AoURN in a Nutshell	74
5.1.	<i>Online Video Store</i>	74
5.2.	<i>Logging Concern (Scenario)</i>	75
5.3.	<i>Logging Concern (Goals)</i>	77
5.4.	<i>Authentication Concern (Scenario)</i>	81
5.5.	<i>Authentication Concern (Goals)</i>	83
5.6.	<i>Communication Concern</i>	85
5.7.	<i>Concern Interactions</i>	89
5.8.	<i>Summary</i>	91
Chapter 6.	Aspect-Oriented User Requirements Notation	92
6.1.	<i>Overview</i>	92
6.1.1	Join Point Model	93
6.1.2	Aspectual Properties.....	93
6.1.3	Pointcut Expressions	93
6.1.4	Composition Rules	94
6.1.5	Concerns	95

6.2.	<i>Aspect-oriented GRL</i>	98
6.2.1	Join Point Model	98
6.2.2	Aspectual Properties.....	98
6.2.3	Pointcut Expressions	99
6.2.4	Composition Rules	100
6.3.	<i>Aspect-oriented UCM</i>	102
6.3.1	Join Point Model	102
6.3.2	Aspectual Properties.....	103
6.3.3	Pointcut Expressions	103
6.3.4	Composition Rules	109
6.4.	<i>Navigating AoURN Models with AoViews</i>	114
6.5.	<i>Advanced Features of AoURN</i>	116
6.5.1	Overview	116
6.5.2	Anything and Anytype Pointcut Elements.....	118
6.5.3	Pointcut Variables	121
6.5.4	Tunnel Aspect Markers and Conditional Aspect Markers.....	125
6.5.5	Local Start and End Points.....	129
6.5.6	Interleaved Composition Rules	134
6.6.	<i>AoURN Metamodel</i>	137
6.6.1	Summary of Concrete Syntax of AoURN	138
6.6.2	Summary of Abstract Syntax of AoURN	138
6.7.	<i>Examples of AoURN Models</i>	143
6.7.1	AoUCM Model for Reservation System	143
6.7.2	AoURN Model for Online News System	147
6.7.3	AoUCM Model for Electronic Voting System	151
6.8.	<i>Relationship of Goal and Scenario Concerns</i>	154
6.9.	<i>Summary</i>	160
Chapter 7. Composition of AoURN Models		164
7.1.	<i>Matching Algorithm</i>	165
7.1.1	Syntax-Based Matching.....	166
7.1.2	Enhanced Matching Based on Semantics	174
7.2.	<i>Composition Algorithm</i>	181
7.2.1	Insertion of AoGRL Aspect Markers	184
7.2.2	Insertion of AoUCM Aspect Markers – Overview	188
7.2.3	Insertion of AoUCM Aspect Markers – Step 1.....	189
7.2.4	Insertion of AoUCM Aspect Markers – Step 2.....	190
7.2.5	Insertion of AoUCM Aspect Markers – Step 3.....	191
7.2.6	Conditional Aspect Markers	191
7.2.7	Tunnel Aspect Markers (Replacement).....	193
7.2.8	Pointcut Variables	195
7.2.9	Insertion Points Before Start Points or After End Points.....	197
7.2.10	Conflicting Aspects	198
7.2.11	URN Model of Composed System.....	199
7.3.	<i>Advanced Features of the Composition Algorithm</i>	200

7.3.1	Interleaved Composition.....	201
7.3.2	Composition of Semantics-Based Matching Results.....	205
7.4.	<i>Beyond Matching and Composition.....</i>	209
7.5.	<i>Summary.....</i>	211
Chapter 8. Qualitative Assessment of AoURN.....		213
8.1.	<i>Goal-Based Approaches to AORE.....</i>	213
8.2.	<i>Qualitative Factors</i>	216
8.3.	<i>Scenario/Use Case-Based Approaches to AORE</i>	218
8.4.	<i>Summary.....</i>	225
Chapter 9. Quantitative Assessment of AoURN		227
9.1.	<i>Overview of the AoURN Modeling Process</i>	228
9.2.	<i>Car Crash Crisis Management System</i>	229
9.2.1	Identification of Concerns (P1).....	230
9.2.2	Model Functional Concerns with UCM (P2 – modified)	232
9.2.3	Model Functional Concerns with AoUCM (P2)	235
9.2.4	The Concern Interaction Graph (P5).....	242
9.2.5	Model Non-Functional Concerns with UCM and AoUCM (P3 – modified).....	244
9.2.6	Model Stakeholder Concerns with GRL and AoGRL (P4 – modified).....	250
9.3.	<i>YKeyK System</i>	254
9.4.	<i>Quantitative Metrics</i>	255
9.5.	<i>Comparison of URN and AoURN Models.....</i>	258
9.5.1	Car Crash Crisis Management System	258
9.5.2	YKeyK System	264
9.6.	<i>Summary.....</i>	267
Chapter 10. Implementation.....		270
10.1.	<i>Prototype Overview</i>	270
10.2.	<i>Feature List for Complete AoURN Modeling Environment.....</i>	275
10.3.	<i>Summary.....</i>	276
Chapter 11. Conclusions		277
11.1.	<i>Contributions</i>	277
11.2.	<i>Future work</i>	278
Glossary		282
References.....		289
Appendix A: URN Metamodel (Baseline Version).....		303

Appendix B: UCM 2.0 Traversal Mechanism	309
Appendix C: BNF for Name Expressions	315
Appendix D: Matching Algorithm	316
Appendix E: Composition Algorithm	321

List of Figures

Figure 1	Basic Elements of GRL Notation	12
Figure 2	Example of a GRL Model: Tiny Online Business	13
Figure 3	Basic Elements of UCM Notation	16
Figure 4	Connecting Stubs, Plug-in Maps, and Paths.....	17
Figure 5	Example of a UCM Model: Tiny Telephone System	18
Figure 6	Regular (left) and Local (right) Start/End Points.....	28
Figure 7	URN Metamodel Extensions: Deferred Choice	29
Figure 8	URN Metamodel Extensions: local and waitType.....	30
Figure 9	Synchronizing Stub.....	31
Figure 10	URN Metamodel Extensions: Synchronizing Stub.....	32
Figure 11	Singletons and Map Instances	34
Figure 12	One Instance for Each Group of Stubs.....	35
Figure 13	URN Metamodel Extensions: Instances of UCM Maps	36
Figure 14	Flattened Dynamic and Synchronizing Stubs.....	37
Figure 15	Instances and Synchronizing Stubs.....	38
Figure 16	Components on Parent (left) and Plug-in (right) Maps.....	41
Figure 17	Plug-in Bindings for Components	42
Figure 18	Plug-in Bindings for Multiple Components	42
Figure 19	URN Metamodel Extensions: Component Plug-in Bindings	43
Figure 20	Plug-in Bindings for Responsibilities	44
Figure 21	URN Metamodel Extensions: Responsibility Plug-in Bindings.....	44
Figure 22	Original Failure Points and Aborts	45
Figure 23	Failure Point and Failure Path	47
Figure 24	Failure Point and Abort Path	48
Figure 25	Scoping Mechanism for Abort Start Points.....	48
Figure 26	URN Metamodel Extensions: Cancellations and Exceptions	49
Figure 27	URN Metamodel Extensions: Scenario Failures	51
Figure 28	Baseline URN Metamodel with Extensions for UCM 2.0.....	52
Figure 29	Baseline URN Scenario Metamodel with Extensions for UCM 2.0.....	54
Figure 30	Basic Control Flow Patterns (Group 1).....	56
Figure 31	Advanced Branching and Synchronization Patterns – Part I (Group 2)	57
Figure 32	Advanced Branching and Synchronization Patterns – Part II (Group 2)	60
Figure 33	Multiple Instance Patterns (Group 3).....	61
Figure 34	State-Based Patterns – Part I (Group 4)	63
Figure 35	State-Based Patterns – Part II (Group 4).....	65
Figure 36	Cancellation and Force Completion Patterns – Part I (Group 5).....	66
Figure 37	Cancellation and Force Completion Patterns – Part II (Group 5).....	67
Figure 38	Iteration Patterns (Group 6).....	68
Figure 39	Trigger Patterns (Group 8)	69

Figure 40	Two Main Scenarios of the Online Video Store System	74
Figure 41	Step by Step Modeling of the Logging Concern (Scenarios).....	75
Figure 42	Impact of Logging Concern on OVS System (Scenario).....	76
Figure 43	AoView of Logging Concern Applied to OVS System (Scenario)	77
Figure 44	Goal Models of the Online Video Store System.....	78
Figure 45	Step by Step Modeling of the Logging Concern (Goals).....	79
Figure 46	Impact of Logging Concern on OVS System (Goals)	80
Figure 47	Detailed View of Logging Concern Applied to OVS System (Goals)	80
Figure 48	Initial and Improved Model of the Authentication Concern (Scenarios).....	82
Figure 49	AoView of Authentication Concern Applied to OVS System (Scenario)	83
Figure 50	Modeling the Authentication Concern (Goals).....	84
Figure 51	AoViews of Authentication Concern Applied to OVS System (Goals)	85
Figure 52	Intent of Communication Concern.....	86
Figure 53	Modeling the Communication Concern	87
Figure 54	Impact of Communication Concern on OVS System	88
Figure 55	AoViews of Communication Concern Applied to OVS System.....	88
Figure 56	Three Concerns Applied to OVS System.....	89
Figure 57	Interaction of Encryption and Authentication Concerns.....	90
Figure 58	Concern Interaction Graph for OVS System.....	91
Figure 59	Concern Interaction Graph	96
Figure 60	URN Metamodel Extensions: Concern	97
Figure 61	GRL Join Point Model	98
Figure 62	Basic Elements of AoGRL Notation.....	100
Figure 63	UCM Join Point Model	102
Figure 64	Pointcut Expressions of AoUCM Notation	104
Figure 65	Comparing Pointcut Expressions for AoUCM and AoGRL	105
Figure 66	Examples of Visual Pointcut Expressions.....	105
Figure 67	Examples of Pointcut Maps.....	106
Figure 68	Location and Naming of Start Points on Pointcut Maps and Base Maps ...	108
Figure 69	URN Metamodel Extensions: Pointcut Stub	108
Figure 70	Composition Rules of AoUCM Notation.....	110
Figure 71	Examples of Composition Rules.....	111
Figure 72	Aspect Markers of AoUCM Notation and Composed System.....	112
Figure 73	URN Metamodel Extensions: Pointcut Stub and Aspect Stub	113
Figure 74	Navigating AoGRL Models with Aspect Markers and AoViews	115
Figure 75	Navigating AoUCM Models with Aspect Markers and AoViews	116
Figure 76	Allowing Variations with the Anything Pointcut Element	119
Figure 77	Location of Anything Pointcut Element on Pointcut Maps.....	119
Figure 78	Allowing Variations with the Anytype Pointcut Element.....	120
Figure 79	URN Metamodel Extensions: Anything Pointcut Element.....	121
Figure 80	Reuse of UCM Model Elements with Variables	122
Figure 81	Matching of Variables.....	123
Figure 82	Loop Composition.....	128
Figure 83	URN Metamodel Extensions: Aspect Stub	129
Figure 84	Plug-in Bindings for Aspect Markers and Candidates on Aspect Maps.....	130
Figure 85	Local Start and End Points on Aspect Maps – Part I.....	131

Figure 86	Local Start and End Points on Aspect Maps – Part II.....	132
Figure 87	Local Start and End Points on Aspect Maps – Part III	133
Figure 88	Movie Points Scenario of the Online Video Store System	135
Figure 89	Combined Order Movie and Movie Points Scenarios.....	135
Figure 90	Modeling the Movie Points Concern	136
Figure 91	AoViews of Movie Points Concern Applied to OVS System.....	137
Figure 92	Summary of AoURN Elements	138
Figure 93	URN Metamodel with Extensions for AoURN	139
Figure 94	Types of Stubs	140
Figure 95	Metadata for AoURN.....	141
Figure 96	Basic UCM Model for Reservation Use Cases.....	144
Figure 97	Advanced UCM Model for Reservation Use Cases – First Attempt	144
Figure 98	Advanced UCM Model for Reservation Use Cases – Second Attempt.....	145
Figure 99	AoUCM Model for Reservation Use Cases	146
Figure 100	AoGRL Model for Online News System	148
Figure 101	AoUCM Model for Online News System – Part I.....	150
Figure 102	AoUCM Model for Online News System – Part II.....	150
Figure 103	UCM Model for Reporting Use Case of Electronic Voting System.....	151
Figure 104	AoUCM Model for Reporting Use Case of Electronic Voting System.....	152
Figure 105	Goal Model of YKeyK System.....	155
Figure 106	AoGRL Model of YKeyK System.....	157
Figure 107	UC001 Visit Car Park – Enter Car Park Scenario of YKeyK System.....	158
Figure 108	UC001 Visit Car Park – Exit Car Park Scenario of YKeyK System.....	158
Figure 109	UC002 Search Car Use Case of YKeyK System.....	159
Figure 110	Relationship of Goal and Scenario Concerns	160
Figure 111	Domain Model for the Matching Algorithm	165
Figure 112	Mappings Between Pointcut Elements and Base Model.....	168
Figure 113	Contradictory Mappings.....	169
Figure 114	Matching Component Hierarchies	172
Figure 115	Contradictory Mappings and the Anything Pointcut Element.....	173
Figure 116	Types of Semantic Equivalences in UCM Models	175
Figure 117	GRL Decomposition Chains.....	176
Figure 118	Semantics-Based Matching of Static Stubs	178
Figure 119	Flattened Dynamic Aspect Marker	180
Figure 120	Order of Concern Composition.....	183
Figure 121	Order of Concern Execution.....	184
Figure 122	Composing AoGRL Concerns	185
Figure 123	Updated Metadata for AoGRL Composition.....	187
Figure 124	Composing AoGRL Concerns – Special Case	188
Figure 125	Composing AoUCM Concerns	190
Figure 126	Composing AoUCM Concerns – Conditional.....	192
Figure 127	Composing AoUCM Concerns – Replacement	194
Figure 128	Composing AoUCM Concerns – Variables	196
Figure 129	Composition with Named Start and End Points	197
Figure 130	Simple Interleaved Composition	202
Figure 131	Interleaved and Loop Composition.....	204

Figure 132	Interleaved and Alternative Composition.....	204
Figure 133	Composition for Semantics-Based Matching Results.....	206
Figure 134	Composition with Shared Plug-in Maps	207
Figure 135	Composition with Lost Hierarchies	208
Figure 136	Updated Metadata for AoUCM Composition	209
Figure 137	Multiple Matches of Pointcut Map	211
Figure 138	UCM: Resolve Crisis Use Case	233
Figure 139	UCM: Resolve Crisis Use Case – ExecuteMission Plug-in Map	234
Figure 140	AoUCM: Coordination Concern – Resolve Crisis Use Case	236
Figure 141	AoUCM: Resolve Crisis Use Case – ExecuteMission Plug-in Map	236
Figure 142	AoUCM: Recommend Strategies Concern	237
Figure 143	AoUCM: Communication Concern – New Crisis and Mission Info	238
Figure 144	AoUCM: Resource Management Concern.....	239
Figure 145	AoUCM: Resource Management Concern – AoViews	239
Figure 146	AoUCM: Coordination Concern – Helicopter Transport Mission	241
Figure 147	AoUCM: Concern Interaction Graph.....	243
Figure 148	AoUCM: Resolve Crisis Use Case with Aspect Markers	244
Figure 149	AoUCM: Availability Concern.....	247
Figure 150	AoUCM: Mobility Concern – Infrastructure.....	248
Figure 151	AoUCM: Mobility Concern – Infrastructure – AoView	248
Figure 152	AoUCM: Safety Concern – Weather Information System.....	249
Figure 153	GRL: Stakeholder Dependencies	250
Figure 154	GRL: Impact of Use Cases on Stakeholders	251
Figure 155	GRL: Availability NFR Concern.....	253
Figure 156	AoGRL: Availability NFR Concern	253
Figure 157	UCM Model for the Resolve Crisis Use Case with All NFR Concerns	258
Figure 158	jUCMNav: Concern Management	271
Figure 159	jUCMNav: Specification of Movie Points Concern	272
Figure 160	jUCMNav: Composition of Movie Points Concern.....	273
Figure 161	Abstract Syntax: URN Top Level.....	303
Figure 162	Abstract Syntax: GRL	304
Figure 163	Abstract Syntax: UCM Core Overview	305
Figure 164	Abstract Syntax: UCM Scenarios Overview	306
Figure 165	Concrete Syntax: URN Top Level.....	306
Figure 166	Concrete Syntax: GRL	307
Figure 167	Concrete Syntax: UCM	308

List of Tables

Table 1	Instances and Synchronizing Stubs.....	38
Table 2	Support of UCM 2.0 Notation for Workflow Patterns.....	70
Table 3	Summary of Assessment of all 43 Workflow Patterns	71
Table 4	Comparison of UCM 2.0, BPMN, UML 2.0 AD, and BPEL4WS	71
Table 5	Matching Rules for Start Points on Pointcut Maps and Base Maps	107
Table 6	Comparing Scenario-Based Approaches to AORE	223
Table 7	Assessment of Resolve Crisis UCM Model Against Identified Concerns..	235
Table 8	Mapping for Adapted Metrics	256
Table 9	Definition of Metrics.....	257
Table 10	CCCMS Results of the Separation of Concerns Metrics	261
Table 11	CCCMS Results of the Coupling, Cohesion, and Size Metrics.....	261
Table 12	CCCMS Results of the Task-Based Evaluation	263
Table 13	YKeyK Results of the Metrics.....	265
Table 14	Features for Complete AoURN Modeling Environment	275
Table 15	Requirements for the UCM 2.0 Traversal Mechanism	309
Table 16	Requirements for UCM 2.0 Scenario Definitions	313
Table 17	Glossary of Terms Used in Requirements.....	314

List of Acronyms

Acronym	Definition
AD	Activity Diagram
AHP	Analytic Hierarchy Process
AoGRL	Aspect-oriented and Goal-oriented Requirement Language
AOM	Aspect-Oriented Modeling
AOP	Aspect-Oriented Programming
AORE	Aspect-Oriented Requirements Engineering
AOSD	Aspect-Oriented Software Development
AoUCM	Aspect-oriented Use Case Maps
AoURN	Aspect-oriented User Requirements Notation
APS	Activity Pattern Specification
ASM	Abstract State Machine
AvNME	Average Number of Model Elements
BNF	Backus–Naur Form
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPMN	Business Process Modeling Notation
CBMU	Coupling between Model Units
CCCMS	Car Crash Crisis Management System
CDKME	Concern Diffusion over Key Model Elements
CDME	Concern Diffusion over Model Elements
CDMU	Concern Diffusion over Model Units
CIG	Concern Interaction Graph
CTS	Clocked Transition Systems
EA	Early Aspects
EPC	Event-driven Process Chain
EVS	Electronic Voting System
GQM	Goal-Question-Metric
GRL	Goal-oriented Requirement Language
IPS	Interaction Pattern Specification
ITU	International Telecommunication Union
KME	Key Model Element
LCOKME	Lack of Cohesion in Key Model Elements
LOC	Lines of Code
LOTOS	Language Of Temporal Ordering Specification
MATA	Modeling Aspects Using a Transformation Approach
MDSOC	Multi-dimensional Separation of Concerns
ME	Model Element
MSC	Message Sequence Chart

Acronym	Definition
MU	Model Unit
NFR	Non-Functional Requirement
NME	Number of Model Elements
OCL	Object Constraint Language
OVS	Online Video Store
SDL	Specification and Description Language
SMPS	State Machine Pattern Specification
SPL	Software Product Lines
TA	Timed Automata
UCM	Use Case Maps
UCS	Universal Character Set
UML	Unified Modeling Language
URN	User Requirements Notation
VS	Vocabulary Size
WCP	Workflow Control-Flow Pattern
XML	Extensible Markup Language
XPDL	XML Process Definition Language
YAWL	Yet Another Workflow Language
YKeyK	Your Key Knows

Chapter 1. Introduction

The introductory chapter summarizes the motivation for this research, concisely defines the research hypothesis in one sentence, lists all contributions and publications that are based on this research, and outlines the content of this document.

1.1. Motivation

By the end of the 1990s, Aspect-Oriented Programming (AOP) [68] allowed software engineers to better encapsulate, at the implementation level, crosscutting concerns (i.e., aspects) which are notoriously difficult to modularize with a single dominant modularization technique alone (e.g., with object-oriented concepts). During the last decade, the research community has shifted its emphasis more towards Early Aspects (EA) [45] by investigating ways of addressing crosscutting concerns in requirements and design models.

Two of the most common requirements engineering models are goal-oriented and scenario-based models. The User Requirements Notation (URN) [18][59][144] is the first, and currently only, standard that combines goal and scenario requirements models in one language. URN is a standardization effort of the International Telecommunication Union (ITU-T Z.150 Series) and contains two complementary, graphical notations. Goals are modeled with the help of the Goal-oriented Requirement Language (GRL) [60] whereas scenarios are modeled with Use Case Maps (UCM) [60]. To date and to the best of our knowledge, no framework combining aspects, goals, and scenarios exists in the context of a requirements modeling standard.

The Aspect-oriented URN (AoURN) presented in this work aims to extend URN with aspect concepts to better manage crosscutting concerns in goal and scenario models. AoURN unifies goal-oriented, scenario-based, and aspect-oriented concepts in one framework, and consists of Aspect-oriented Use Case Maps (AoUCM) and the Aspect-oriented and Goal-oriented Requirement Language (AoGRL). A prerequisite for extending URN with aspect-oriented concepts is for URN to be a precisely defined modeling

language. While aspects may be applied based only on syntactical information about a model, a technique that utilizes the deeper semantics of a modeling notation results in a much more powerful approach. To that effect, the semantics of URN must be clarified, and the abstract and concrete syntax of URN must be extended to ensure that necessary and useful semantic variations can be fully expressed with the notation.

While this work focuses on the URN notation in the context of requirements engineering activities, the URN notation is applicable to a much larger set of activities, for example but not limited to design and testing activities. This also applies to AoURN. Therefore, the advantages of AoURN discussed in this work may also be applicable to these other activities.

1.2. Research Hypothesis

The goal of this research is to create a URN-based requirements modeling framework that provides better encapsulation for all types of concerns at the goal and scenario-level regardless of whether concerns crosscut or not. At the same time, changes to URN should be kept to a minimum in order to ensure that requirements engineers can continue working with familiar notations. Because there is a strong overlap between goals as well as scenarios on the one side and crosscutting concerns on the other side, such concerns are encapsulated across model types with AoURN, further bridging the gap between goals and scenarios which describe how a goal is achieved. EA research, on the other hand, benefits from a standardized way of modeling crosscutting concerns with AoURN.

*AoURN improves the modularity, reusability, scalability,
and maintainability of URN models.*

With these improvements, requirements engineers should see higher model quality, enhanced productivity, and a better alignment of model units with real-world concepts such as security, performance, transaction control, persistence, and availability, to name a few. Furthermore, these improvements may result in a greater specialization in areas related to these real-world concepts, because clearly separated models are now available.

1.3. Thesis Contributions

The contributions of this research are listed in this section in order of importance. References to the section discussing the contribution are given.

- Unification of goal-oriented, scenario-based, and aspect-oriented concepts in one framework for requirements engineering (Chapter 6, Chapter 7) – this unification requires:
 - the extension of the abstract syntax, the concrete syntax, and the semantics of URN with aspect-oriented concepts, and
 - the clarification of the relationship between goal aspects and scenario aspects within AoURN.
- Flexible and exhaustive composition of aspects that exploits semantic equivalences of URN models and is limited only by the expressive power of URN itself as opposed to a particular composition language (Chapter 7)
 - The composition and visualization of aspects is based on the notion of dynamic stubs, i.e., a notational element of UCM that may possibly be generalized to other languages.
- Clarification of the semantics of URN as semantics-based composition of aspects requires precise language semantics (Chapter 3, Chapter 4)
- Evaluation of AoURN (Chapter 8, Chapter 9) – this evaluation requires
 - a qualitative assessment of AoURN with respect to desirable properties of aspect-oriented requirements languages,
 - a quantitative assessment of AoURN based on case studies and metrics for aspect-oriented requirements models adapted for URN and AoURN, and
 - proof of concept by providing tool support for AoURN.

Future work identifies expanding AoURN concepts to the analysis capabilities of URN and investigating the applicability of advanced URN research to AoURN, exploring the connections of AoURN with aspect-oriented models at different levels of abstraction, investigating the usability of AoURN through empirical studies, strengthening the formal foundation of URN (and hence AoURN), and incorporating further AoURN concepts into the official URN standard.

1.4. Publications Based on Thesis

All publications based on this research are listed in this section in chronological order.

The first author is the main author.

Book Chapters and Refereed Journals:

- Mussbacher, G., Amyot, D., and Weiss, M.: “Visualizing Early Aspects with Use Case Maps”. Rashid, A. and Aksit, M. (Editors), *Transactions on Aspect-Oriented Software Development III*, Springer, LNCS 4620, pp. 105–143 (November 2007)
- Mussbacher, G., Whittle, J., and Amyot, D.: “Modeling and Detecting Semantic-Based Interactions in Aspect-Oriented Scenarios”. *Requirements Engineering Journal (REJ)*, Springer, 15(2), pp. 197–214 (2010)
- Pourshahid, A., Mussbacher, G., Amyot, D., and Weiss, M.: “Toward an Aspect-Oriented Framework for Business Process Improvement”. *International Journal of Electronic Business (IJEB)*, Inderscience Publishers, 8(3), pp. 233–259 (2010)
- Mussbacher, G., Amyot, D., Araújo, J., and Moreira, A.: “Requirements Modeling with the Aspect-oriented User Requirements Notation (AoURN): A Case Study”. Katz, S., Mezini, M., and Kienzle, J. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) VII*, Springer, LNCS 6210, pp. 23–68 (2010)
- Becha, H., Mussbacher, G., and Amyot, D.: “Modeling and Analyzing Non-Functional Requirements in Service Oriented Architecture with the User Requirements Notation”. Milanovic, N. (Editor), *Non-functional Properties in Service Oriented Architecture: Requirements, Models and Methods*, IGI Global (to appear)

Refereed Proceedings:

- Mussbacher, G., Amyot, D., and Weiss, M.: “Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps”. *International Workshop on Requirements Engineering Visualization (REV 2006)*, Minneapolis, Minnesota, USA (September 2006)
- Mussbacher, G.: “Evolving Use Case Maps as a Scenario and Workflow Description Language”. *10th Workshop on Requirements Engineering (WER’07)*, Toronto, Canada, pp. 56–67 (May 2007); http://www.inf.puc-rio.br/~wer/WERpapers/artigos/artigos_WER07/Hwer07-mussbacher.pdf (accessed August 2010)

- Mussbacher, G., Amyot, D., Araújo, J., Moreira, A., and Weiss, M.: “Visualizing Aspect-Oriented Goal Models with AoGRL”. *2nd International Workshop on Requirements Engineering Visualization (REV 2007)*, New Delhi, India (October 2007)
- Mussbacher, G., Amyot, D., Whittle, J., and Weiss, M.: “Flexible and Expressive Composition Rules with Aspect-oriented Use Case Maps (AoUCM)”. *10th International Workshop on Early Aspects (EA 2007)*, Vancouver, Canada (March 13, 2007). Moreira, A. and Grundy, J. (Editors), *Early Aspects: Current Challenges and Future Directions*, Springer, LNCS 4765, pp. 19–38 (December 2007)
- Mussbacher, G. and Amyot, D.: “Assessing the Applicability of Use Case Maps for Business Process and Workflow Description”. *3rd International MCEtech Conference on eTechnologies*, Montreal, Canada, IEEE Computer Society Press, pp. 219–222 (January 23-25, 2008)
- Mussbacher, G., Amyot, D., Araújo, J., and Moreira, A.: “Modeling Software Product Lines with AoURN”. *Early Aspects Workshop: Early Aspects and Software Product Lines (EA-AOSD’08)*, Brussels, Belgium, ACM Digital Library (March 31, 2008)
- Mussbacher, G.: “Aspect-Oriented User Requirements Notation: Aspects in Goal and Scenario Models”. Giese, H. (Editor), *Models in Software Engineering: Workshops and Symposia at MODELS 2007*, Springer, LNCS 5002, pp. 305–316 (August 2008)
- Amyot, D. and Mussbacher, G.: “Development of Telecommunications Standards and Services with the User Requirements Notation”. *Workshop on ITU System Design Languages 2008*, Geneva, Switzerland (September 15-16, 2008) [not refereed]
- Mussbacher, G., Whittle, J., and Amyot, D.: “Towards Semantic-Based Aspect Interaction Detection”. *1st International Workshop on Non-functional System Properties in Domain Specific Modeling Languages (NFPinDSML2008)*, Toulouse, France (September 28, 2008)
- Pourshahid, A., Mussbacher, G., Amyot, D., and Weiss, M.: “An Aspect-Oriented Framework for Business Process Improvement”. *4th International MCEtech Conference on eTechnologies*, Ottawa, Canada (May 4-6, 2009). Babin, G., Kropf, P., and Weiss, M. (Editors), *E-Technologies: Innovation in an Open World*, Springer, LNBIP 26, pp. 290–305 (2009)

- Mussbacher, G. and Amyot, D.: “Heterogeneous Pointcut Expressions”. *Early Aspects Workshop at ICSE'09*, Vancouver, Canada, IEEE Computer Society Press, pp. 8–13 (May 18, 2009)
- Mussbacher, G. and Amyot, D.: “On Modeling Interactions of Early Aspects with Goals”. *Early Aspects Workshop at ICSE'09*, Vancouver, Canada, IEEE Computer Society Press, pp. 14–19 (May 18, 2009)
- Mussbacher, G., Amyot, D., and Whittle, J.: “Semantic-Based Aspect Interaction Detection with Goal Models (Position Paper)”. *10th International Conference on Feature Interactions (ICFI 2009)*, Lisbon, Portugal (June 11-12, 2009). Nakamura, M. and Reiff-Marganiec S. (Editors), *Feature Interactions in Software and Communication Systems X*, IOS Press, pp. 176–182 (2009)
- Mussbacher, G., Amyot, D., Weigert, T., and Cottenier, T.: “Feature Interactions in Aspect-Oriented Scenario Models”. *10th International Conference on Feature Interactions (ICFI 2009)*, Lisbon, Portugal (June 11-12, 2009). Nakamura, M. and Reiff-Marganiec S. (Editors), *Feature Interactions in Software and Communication Systems X*, IOS Press, pp. 75–90 (2009)
- Mussbacher, G., Whittle, J., and Amyot, D.: “Semantic-Based Interaction Detection in Aspect-Oriented Scenarios”. *17th IEEE International Requirements Engineering Conference (RE'09)*, Atlanta, Georgia, USA, IEEE Computer Society Press, pp. 203–212 (September 2009) (Acceptance rate: 21%)
- Mussbacher, G. and Amyot, D.: “Extending the User Requirements Notation with Aspect-oriented Concepts”. *14th SDL Forum (SDL 2009)*, Bochum, Germany (September 2009). Reed, R., Bilgic, A., Gotzhein, R. (Editors), *SDL 2009: Design for Motes and Mobiles*, Springer, LNCS 5719, pp. 115–132 (2009)
- Mussbacher, G., Amyot, D., and Whittle, J.: “Refactoring-Safe Modeling of Aspect-Oriented Scenarios”. *ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MODELS 2009)*, Denver, Colorado, USA (October 2009). Schürr, A. and Selic, B. (Editors), *Model Driven Engineering Languages and Systems*, Springer, LNCS 5795, pp. 286–300 (2009) (Acceptance rate: 18%)

- Mussbacher, G., Barone, D. and Amyot, D.: “Towards a Taxonomy of Syntactic and Semantic Matching Mechanisms for Aspect-oriented Modeling”. *6th Workshop on System Analysis and Modelling (SAM 2010)*, Oslo, Norway (October 2010)
- Mosser, S., Mussbacher, G., Blay-Fornarino, M., and Amyot, D.: “From Aspect-oriented Requirements Models to Aspect-oriented Business Process Design Models – An Iterative and Concern-Driven Approach for Software Engineering”. *10th International Conference on Aspect-Oriented Software Development (AOSD 2011)*, Porto de Galinhas, Brazil (March 2011, to appear)

In addition, two tutorials with material from this work were presented at the AOSD’07 and RE’09 conferences.

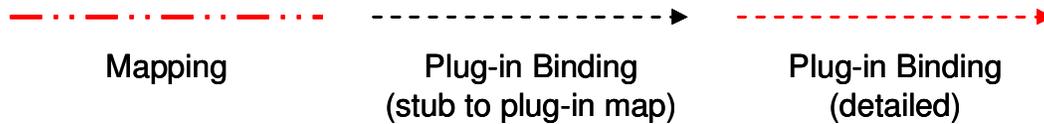
- Mussbacher, G., Araújo, J., Moreira, A., and Whittle, J.: “Aspect-Oriented Requirements Engineering With Scenarios”. *7th International Conference on Aspect-Oriented Software Development*, Brussels, Belgium (March 31 - April 4, 2008)
- Mussbacher, G. and Amyot, D.: “The User Requirements Notation (URN) and Aspects”. *17th IEEE International Requirements Engineering Conference (RE’09)*, Atlanta, Georgia, USA (September 2009)

Furthermore, the latest version of the URN standard [60] includes some of the recommendations from Chapter 3, Chapter 4, and Chapter 6 as indicated in these chapters. Therefore, this work is presented relative to a baseline version of the URN standard (see Appendix A: URN Metamodel (Baseline Version)) that is the same as the latest version but without the enhancement discussed in this work.

1.5. Formatting Conventions

Italic font style indicates important domain-specific terms while the usage of a sans serif font refers to text in figures. Some figures contain additional elements that are not part of the URN or AoURN notation but are added to the figures for further clarity. The red long-dash-dot-dotted lines without arrowheads help illustrate the mapping of the pointcut expression to the base model. The dashed arrows are added to clearly indicate plug-in bindings for UCM models. While black dashed arrows indicate a plug-in map of a stub, red dashed arrows indicate detailed plug-in bindings between a stub and elements on a

plug-in map. Because these mappings and bindings are not part of the official URN and AoURN syntax, URN and AoURN modeling tools are not required to and hence do not display them on URN and AoURN diagrams, respectively, but typically manage these mappings and bindings more concisely through other means.



1.6. Thesis Outline

In the remainder of this work, Chapter 2 gives an overview of URN and aspect concepts. Chapter 3 presents extension to the UCM notation (i.e., UCM 2.0) including an updated URN metamodel to clarify the semantics of UCM and address longstanding issues with the language. Chapter 4 reports on an assessment of the applicability of UCM 2.0 for scenario-based requirements engineering based on workflow patterns. Chapter 5 gives an intuitive, example-based introduction to AoURN. Chapter 6 then describes the core of AoURN and discusses the relationship of aspects in goal and scenario models. Several examples of AoURN models illustrate the use of AoURN. Chapter 7 introduces the matching and composition algorithms of AoURN. The following three chapters report on the validation of AoURN which is based on three pillars: implementation, case studies, and assessments based on qualitative and quantitative factors. Chapter 8 identifies qualitative attributes of aspect-oriented models at the requirements level, gives an overview of related work on EA for goal and scenario models, and assesses AoURN against the related work. Chapter 9 identifies quantitative attributes of aspect-oriented models at the requirements level, describes a case study involving two systems, and assesses the case study models. Chapter 10 discusses the proof-of-concept implementation of AoURN. Last but not least, Chapter 11 concludes the thesis and identifies future work.

Chapter 2. Background

The background chapter gives an overview of the User Requirements Notation (URN) and its two complementary languages, the Goal-oriented Requirement Language (GRL) and Use Case Maps (UCM), based on the version of the URN standard in Appendix A: URN Metamodel (Baseline Version). The chapter concludes with a brief overview of aspect-oriented concepts.

2.1. User Requirements Notation

The User Requirements Notation (URN) [10][18][59][144], a standard of the International Telecommunication Union (ITU-T Z.150 Series), contains two complementary modeling languages for goals and scenarios. The URN language definition [60] was published by ITU-T in November 2008. URN allows for the seamless capture of stakeholders' goals, rationales, and alternative solutions to achieve such goals. The solutions are reasoned about with the Goal-oriented Requirement Language (GRL) and their behavior and structure described in more detail with Use Case Maps (UCM).

In general, URN facilitates the elicitation and specification of requirements for a proposed or an evolving system as well as the analysis and validation of requirements for correctness and completeness. While the static semantics of URN is defined by the URN metamodel [60], natural language descriptions, and constraints, the dynamic aspects of URN are defined by *propagation mechanisms* for GRL model evaluation and by a *path traversal mechanism* for UCM scenario interpretation. The former allows for the comparison of alternatives and facilitates trade-offs among conflicting goals of various stakeholders. The latter is the basis for many advanced applications of UCM, such as scenario highlighting and animation, the generation of MSCs, and the generation of test cases.

URN links (▶) can link any two URN model elements. In particular, links from GRL models to UCM models establish traceability between goal and scenario models in URN. URN is the first and currently only standard to address explicitly, in a graphical

way, and in one unified language goals and scenarios, and the links between them. Furthermore, *metadata* in the form of name/value pairs may be associated with any URN model element. This approach allows for domain-specific extensions to be added to URN and exploited by specialized tool support.

Two editing tools for URN have been developed at the University of Ottawa and Carleton University. The tools make it possible to create, maintain, analyze, and transform URN models. The original UCM Navigator (UCMNAV) [79][142] is only a UCM tool whereas the new Eclipse-based *jUCMNav* tool [62][66][123][124] is a true URN tool that offers GRL modeling in addition to UCM modeling. *jUCMNav* also allows OCL [111] constraints to be defined [11] which, in conjunction with metadata, can be used to define URN profiles for other languages (e.g., *i** [14]). Other tools include *OpenOME* and the *SandriLa SDL* stencil set. *OpenOME*, a requirements engineering tool available from the University of Toronto [114], focuses on general goal and/or agent modeling and allows GRL models to be specified and analyzed in addition to other goal models. SanDriLa Ltd provides a Microsoft Office Visio template for GRL models in their *SandriLa SDL* stencils set [132] that can be used as a simple GRL drawing tool.

URN is a general purpose modeling language for the communication of and reasoning about requirements. URN can be applied to standards development [10] as well as to various types of systems. Over the last decade, GRL and UCM have successfully been used for service-oriented, concurrent, distributed, and reactive systems such as telecommunications systems [9][12][19][57], e-commerce systems [16], agent systems [46], operating systems [28], and health information systems [1][108]. Moreover, URN has been used for business process modeling, monitoring, and alignment [118][119][121][147], for reverse engineering [15][53], for tracking compliance with institutional and governmental policies, regulations, and legislation [48][49], and for the formalization of patterns [89]. Modeling Software Product Lines (SPL) has also been suggested as an application area of URN [94]. Furthermore, UCM and GRL models have already been integrated with requirements management systems [65] and [48], respectively. Many examples of URN models can be found in the publications referenced in this paragraph. An extensive body of research composed of almost 300 publications related to UCM, GRL, and URN is available at the URN Virtual Library [144].

2.1.1 Goal-Oriented Requirement Language

The *Goal-oriented Requirement Language* (GRL) [10][18][60][124][144] is a visual modeling notation for business goals and non-functional requirements (NFRs) of many stakeholders, for alternatives that have to be considered, for decisions that were made, and for rationales that helped make these decisions. Major benefits of GRL over other popular notations include the integration of GRL with the scenario notation UCM, the support for qualitative as well as quantitative attributes, and a clear separation of GRL model elements from their graphical representation, enabling a scalable and consistent representation of multiple views of the same goal model.

By capturing solutions that have been considered and the reasons why one solution is superior to another, costly re-evaluations of the same solutions at a later stage in the software development process may be avoided. Furthermore, if new goals emerge at a later stage or the context for decisions changes, the initial solutions may be re-evaluated to determine the best solution under the new circumstances. By clearly documenting these decisions, uninformed, premature system decisions may be reduced.

GRL supports reasoning about goals and requirements, especially NFRs and quality attributes, as it shows the impact of often conflicting goals and various alternative solutions proposed to achieve the goals. A GRL *strategy* describes a particular configuration of alternatives in the GRL model. An *evaluation mechanism* propagates these low-level decisions regarding alternatives to satisfaction ratings of high-level stakeholder goals and NFRs.

A goal model that can be reused for many application domains is called a *GRL catalogue* (e.g., a goal graph that describes performance issues or a group of features).

GRL combines the Non-Functional Requirements Framework (NFR Framework) [38] and i* framework [160] to support reasoning about goal models. The syntax of GRL (Figure 1) is based on the syntax of the i* framework. A GRL goal graph is a connected graph of *intentional elements* that optionally reside within an *actor boundary*. An *actor* (, e.g., Business Owner, Figure 2) represents a stakeholder of a system. A goal graph shows the high-level business goals and non-functional requirements of interest to a stakeholder and the alternatives for achieving these high-level elements. A goal graph also documents *beliefs* (i.e., rationales) important to the stakeholder. In addition to

beliefs, intentional elements may be softgoals, goals, tasks, and resources. *Softgoals* (◡, e.g., System Security) differentiate themselves from *goals* (◻, e.g., Offer Online Shopping) in that there is no clear, objective measure of satisfaction for a softgoal whereas a goal is quantifiable. In general, softgoals are more related to NFRs, whereas goals are more related to functional requirements. *Tasks* (◁, e.g., Fingerprint) represent solutions to (or *operationalizations* of) goals or softgoals. In order to be achieved or completed, softgoals, goals, and tasks may require *resources* (◻, e.g., Payment) to be available. Actors are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, resources to be available, and softgoals to be satisfied. Goals, softgoals, tasks, resources, and beliefs are intentional because they are used for models that allow answering questions such as why particular behaviors, informational and structural aspects were chosen to be included in the system requirements, what alternatives were considered, what criteria were used to deliberate among alternative options, and what the reasons were for choosing one alternative over the other.

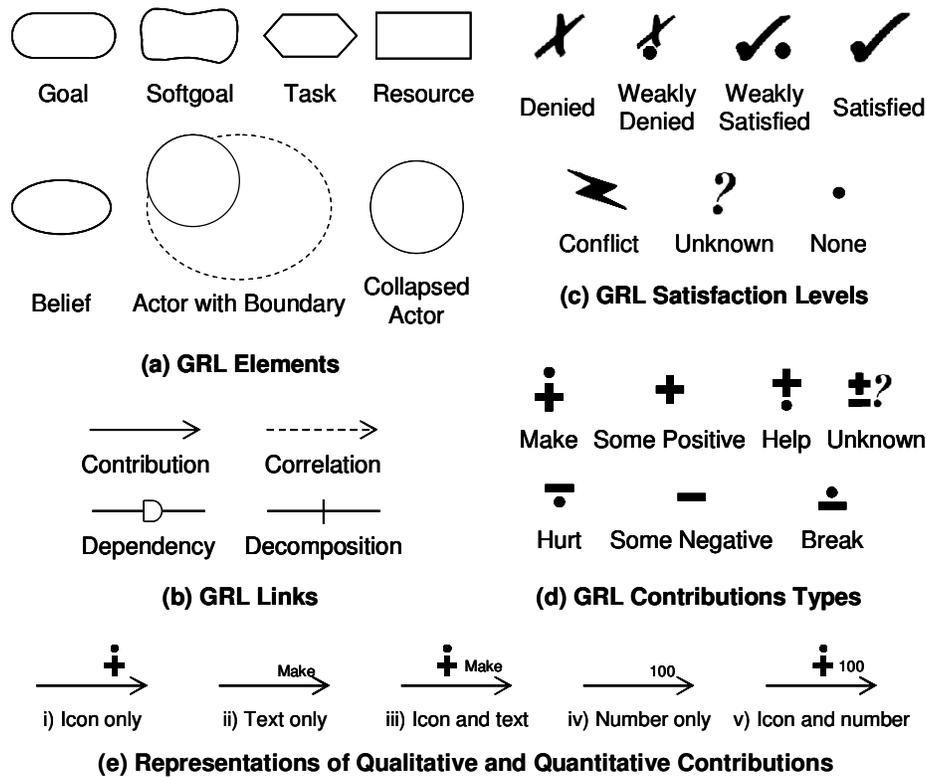


Figure 1 Basic Elements of GRL Notation

Various kinds of *links* connect the elements in a goal graph. Decomposition links allow an element to be decomposed into sub-elements (\vdash , e.g., Access Authorization is decomposed into Authentication and Identification). AND, IOR, as well as XOR decompositions are supported. Contribution links indicate desired impacts of one element on another element (\rightarrow , e.g., Security of Terminal contributes to System Security). A contribution link has a qualitative contribution type (Figure 1.d) or a quantitative contribution, i.e., an integer value between -100 and 100 (Figure 1.e). Correlation links are similar to contribution links, but describe side effects rather than desired impacts (\dashrightarrow , e.g., Password has a side-effect on Cost of Terminal). Finally, dependency links model relationships between actors, i.e., one actor depending on another actor for something (\dashv , e.g., Business Owner depends on Online Shopper for Payment). A more complete coverage of the notation elements is available in [10][18][60][124].

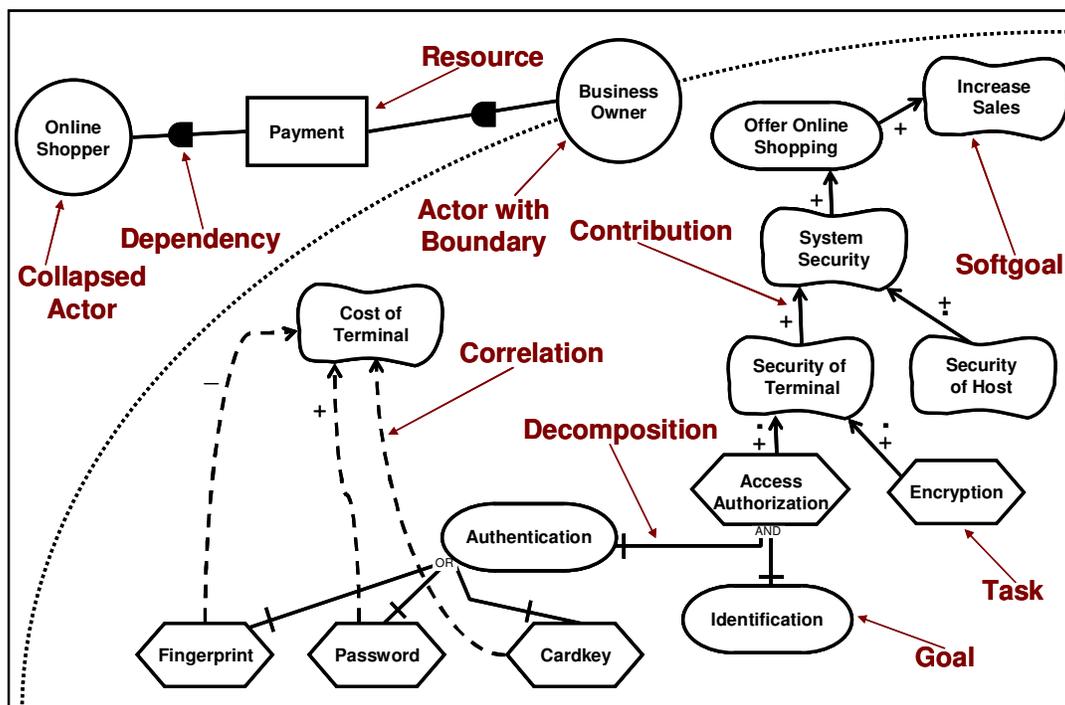


Figure 2 Example of a GRL Model: Tiny Online Business

Figure 2 shows an example of a GRL model that illustrates the concepts discussed in this section. The model of a Tiny Online Business includes two stakeholders, Online Shopper and Business Owner, with a dependency on Payment between them. The Business Owner's goals are shown in more detail. The Business Owner wants to Increase Sales

and is thinking of the option Offer Online Shopping to achieve this sales increase. System Security needs to be considered in this case and an excerpt of the goal tree for security is shown. Security of Host is not further refined. Security of Terminal, on the other hand, is impacted by Encryption and Access Authorization which in turn is decomposed into Authentication and Identification. Three alternatives for Authentication are shown with their side-effects to Cost of Terminal.

From the NFR Framework, GRL borrows the notion of an evaluation mechanism that supports reasoning about the goal graph. The decisions of stakeholders are typically documented in the goal graph by the assignment of satisfaction levels (Figure 1.c) to alternatives (e.g., the chosen alternative is set to Satisfied whereas all other alternatives are set to None or Denied). Based on these initial settings and the various links with various contribution types (Figure 1.d), the satisfaction ratings are propagated to higher-level goals and non-functional requirements of stakeholders. This propagation enables a global assessment of the alternative being studied. GRL keeps track of these initial settings separate from goal graphs in *strategies*. Several strategies may be defined for a goal model, allowing trade-off analyses to be performed by exploring and comparing various configurations of alternatives. GRL also takes into account that not all high-level goals and non-functional requirements are equally important to the stakeholder. Therefore, GRL supports the definition of an *importance* attribute (again quantitative or qualitative) for intentional elements inside actors, which is also taken into account when evaluating strategies for the goal model. A qualitative importance attribute may be set to High, Medium, Low, or None, whereas a quantitative importance attribute is an integer value between 0 and 100.

2.1.2 Use Case Maps

Use Case Maps (UCM) [10][18][31][33][60][144] is a visual scenario notation that focuses on the causal flow of behavior optionally superimposed on a structure of components. UCM depicts the causal interaction of architectural entities while abstracting from message and data details. The goal of the UCM notation is to provide the right degree of formality at the right time in the development process. UCM supports the definition of *scenarios* including pre- and postconditions. A scenario describes a specific path through

the UCM model where only one alternative at any choice point is taken. Given the definition of a scenario or combination of scenarios, a *path traversal mechanism* can highlight the scenario path or transform the scenario into more concrete design notations such as message sequence charts (MSCs). The traversal mechanism effectively turns the scenario definitions into a test suite for the UCM model. The UCM notation enables a seamless transition from the informal to the formal by bridging the modeling gap between goal models and natural language requirements (e.g., use cases) and design, in an explicit and visual way.

UCM is ideally suited for the description of functional requirements and, if desired, high-level design. Paths describe the causal flow of behavior of a system (e.g., one or many use cases). By superimposing paths over components, the architectural structure of a system can be modeled. In general, components describe any kind of structural entity at any abstraction level (e.g., classes or packages but also systems, actors, sub-systems, objects, concerns, aspects, hardware). As many scenarios and use cases are integrated into one combined UCM model of a system, it is possible to use the semi-formal UCM specifications as a basis for further analysis. Undesired interactions between scenarios can be detected [13][77][136][148][149], performance implications can be analyzed [116][117][135][137], testing efforts can be driven based on the UCM model [17], and various architectural alternatives can be analyzed [18][19][147]. Extensions allow UCM to be used for user interface modeling [7] as well as real-time modeling [55][56]. For further information, the reader is referred to the URN Virtual Library [144].

The basic elements of the UCM notation are shown in Figure 3. A more complete coverage of the notation elements is available in [10][18][31][33][60]. A *map* (e.g., Basic Call in Figure 5) contains any number of paths and components. *Paths* express causal sequences and may contain several types of path nodes. Paths start at *start points* (●, e.g., request) and end at *end points* (■, e.g., signal). *Responsibilities* (✕, e.g., verify) describe required actions or steps to fulfill a scenario. *OR-forks* (\neg) (possibly including guarding conditions, e.g., idle) and *OR-joins* (\simeq) are used to show alternatives, while *AND-forks* (\dashv) and *AND-joins* (\dashv) depict concurrency. Loops can be modeled implicitly with OR-joins and OR-forks. As the UCM notation does not impose any nesting constraints, joins and forks may be freely combined and a fork does not need to be followed by a join.

Waiting places (●) and timers (⊖) denote locations on the path where the scenario stops until a condition is satisfied. If an endpoint is connected to a waiting place or a timer, the stopped scenario continues when this end point is reached, i.e., synchronous interaction. Asynchronous, in-passing triggering of waiting places and timers is also possible. A timer may also have a timeout path which is indicated by a zigzag line. End points and start points of paths may be connected to each other as shown in Figure 4.c to indicate simple sequences of paths.

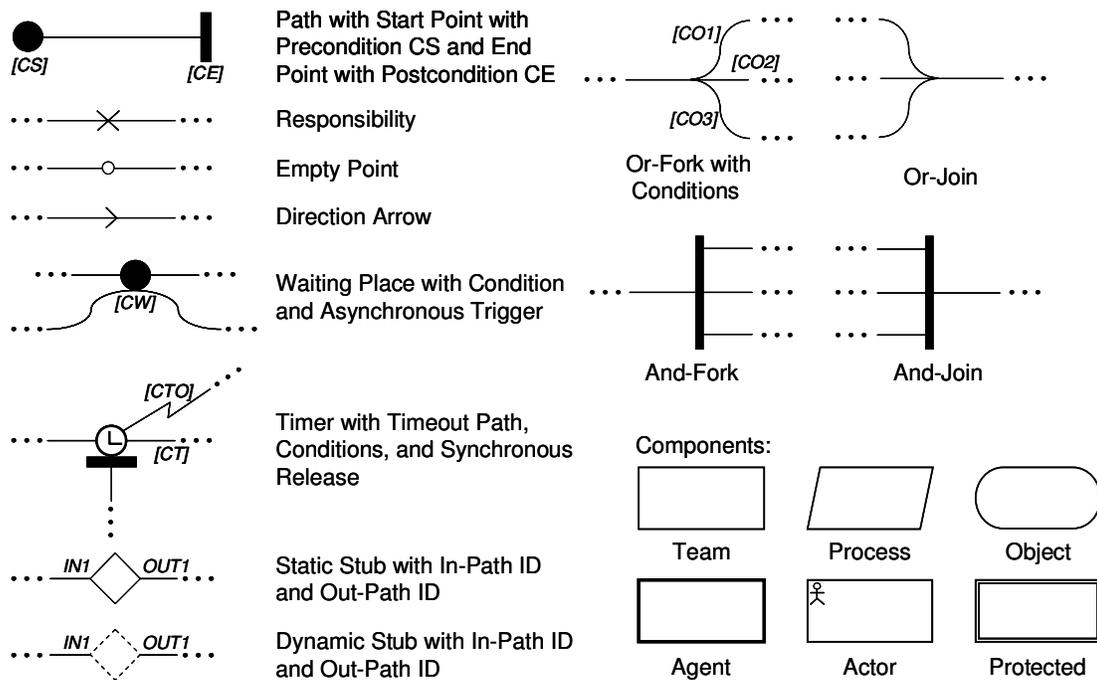


Figure 3 Basic Elements of UCM Notation

UCM models can be decomposed using *stubs* which contain sub-maps called *plug-in maps* (e.g., Teenline). Plug-in maps are reusable units of behavior and structure. *Plug-in bindings* define the continuation of a path on a plug-in map by connecting in-paths and out-paths of a stub with start and end points of its plug-in maps, respectively (Figure 4.a). Stubs without plug-in bindings for start or end points may be shown visually without an in-path or out-path, respectively (Figure 4.b). A stub may be *static* (◇) which means that it can have at most one plug-in map, whereas a *dynamic* stub (◊, e.g., OriginatingFeatures) may have many plug-in maps which may be selected at runtime. A *selection policy* decides which plug-in maps of a dynamic stub to choose at runtime.

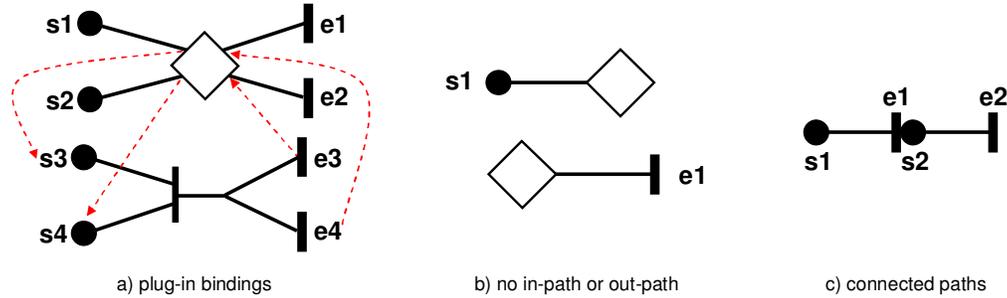


Figure 4 Connecting Stubs, Plug-in Maps, and Paths

Components (\square , e.g., Telephone Switch) are used to specify the structural aspects of a system. Map elements which reside inside a component are said to be bound to the component. Components may contain sub-components and have various types and characteristics. For example, a *protected* component does not allow a second path to enter the component if one path is already executing inside the component and a component of kind *object* does not have its own thread of control while a component of kind *process* does. A component of kind *actor* (\otimes , e.g., Caller) represents someone or something interacting with the system under design. Any kind of component can be a protected component.

Figure 5 shows an example of a UCM model that illustrates the concepts discussed in this section. The model of a Tiny Telephone System consists of three maps, Basic Call, Teenline, and default, and makes use of two components, Telephone Switch and the actor Caller. The scenario starts at request. The dynamic stub OriginatingFeatures contains the default and Teenline plug-in maps, one of which will be chosen depending on whether the caller has subscribed to the Teenline feature. The Teenline feature allows parents to define a time period where a PIN is required to make a phone call. If the caller is not subscribed, then the scenario continues on the default plug-in map and exits the dynamic stub along OUT1 since the end point on the default plug-in map is bound to this out-path. The call is then verified. If the callee is idle, the call information is updated, the callee's phone rings, and the ringback signal is applied to the caller's phone. If the callee is busy, the busy signal is applied to the caller's phone. In both cases, the scenario ends at the signal end point. If the caller is subscribed to the Teenline feature, the scenario continues on the Teenline plug-in map. First, the time is checked and if the Teenline feature is not active, the Teenline feature ends at the success end point. If the Teenline feature is

active, the system waits at the timer for the Caller to enter the PIN. If this occurs before the timer times out, the PIN is checked and if it is valid, the Teenline feature ends at the success end point. The success end point is bound to the OUT1 out-path and the call is verified like before. If the PIN is invalid or if the timer times out before the PIN is entered, the deny signal is applied to the caller's phone and the Teenline feature ends at the fail end point. The fail end point is bound to the OUT2 out-path and the scenario ends at the signal end point.

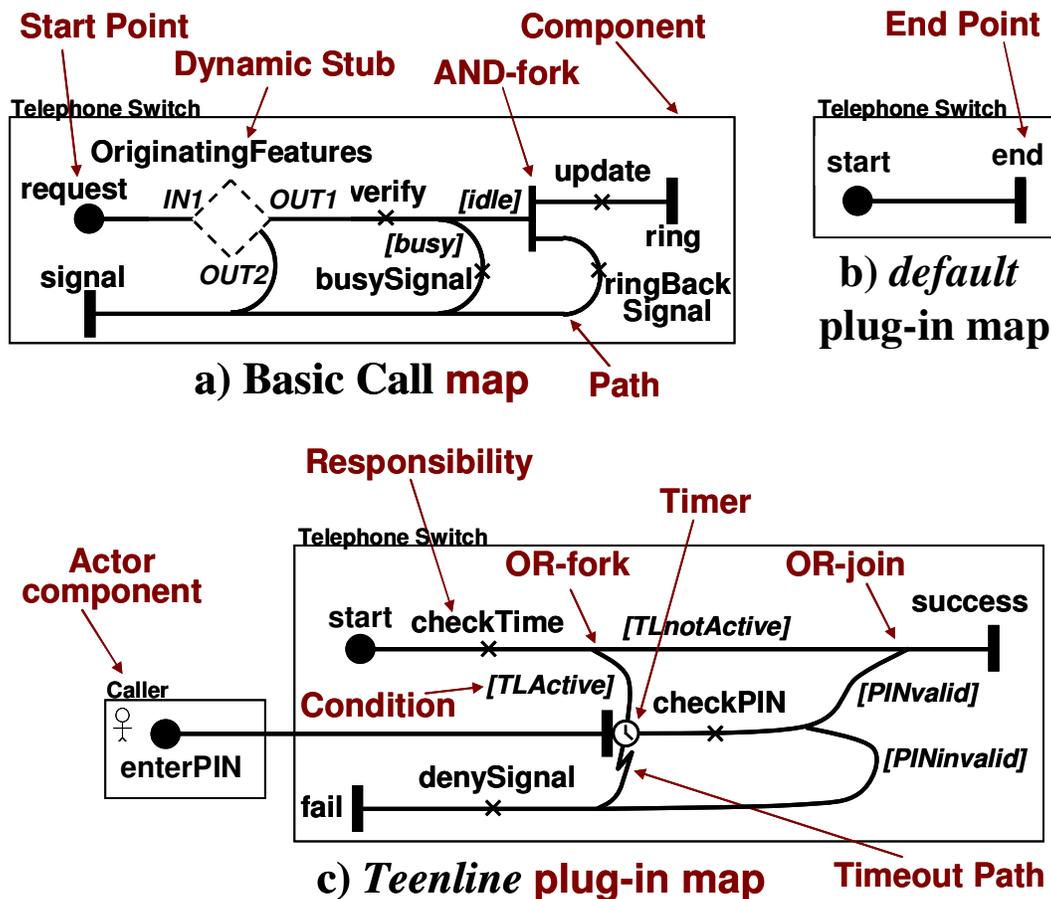


Figure 5 Example of a UCM Model: Tiny Telephone System

UCM may be further formalized with the help of *scenario* definitions. Each scenario specifies its preconditions and postconditions as well as its start points and expected end points. For each choice point in the UCM model (e.g., an OR-fork or a dynamic stub), Boolean expressions are defined for all alternatives. Boolean expressions may also be defined for start points, capturing preconditions of paths. The expressions may contain Boolean, Integer, and Enumeration *scenario variables*. A scenario describes a specific

path through the UCM model by initializing the variables used in the Boolean expressions (only one alternative at a choice point is taken and only the start points with pre-conditions satisfied by the variable initializations are enabled).

One of the jUCMNav tool's features is a *traversal mechanism* which, besides improving the usability of the tool and allowing for a smoother transition to downstream modeling activities, defines more precisely the semantics of UCM. As no standard traversal mechanism existed for UCM before November 2008, the jUCMNav tool's traversal mechanism [62][64][66] was based on a simple but intuitive interpretation of the notational elements for sequences, alternatives, and concurrency in UCM. An OR-fork is an exclusive or, there is no synchronization on an OR-join, an AND-fork denotes strict concurrency, an AND-join requires all incoming branches to arrive before continuation, and the selection policy of a dynamic stub is also an exclusive or. In addition, scenario variables can be evaluated and changed during the traversal of the UCM model. Given a scenario definition indicating start points triggered and initial values of variables, the traversal mechanism is capable of highlighting the scenario in a UCM model or translating it into message sequence charts (MSCs). With the help of the traversal mechanism, the scenario definitions effectively become a test suite for the UCM model. Note that selection policies have since been described in the URN standard with a multiple choice semantics.

UCM shares many characteristics with UML activity diagrams, but UCM offers more flexibility in how sub-diagrams can be connected and how sub-components can be represented. UCM also integrates a simple data model, performance annotations, and a simple action language used for analysis. Activity Diagrams, however, have better support for data flow modeling, object flows, cancellation and exception handling, and a better integration with the rest of UML. UCM, on the other hand, is better integrated with goal-oriented models created with the Goal-oriented Requirement Language (GRL).

2.2. Basic Concepts of Aspect-Oriented Software Development

Separation of concerns is a fundamental software engineering principle to manage complexity, improve reusability, and simplify evolution. While many programming and modeling languages provide good support for partitioning software systems into modules, the

restriction to modularize a software system along a single dimension can make it difficult or impossible to encapsulate concerns that affect many modules in the primary decomposition. About a decade ago, Aspect-Oriented Programming (AOP) [68] introduced a new way of structuring software systems. Based on a more general philosophy called *multi-dimensional separation of concerns* (MDSOC) [39][141], this new type of modularization makes it possible to address problems of object-oriented software engineering that occur because the units of interest to a requirements engineer (i.e., the concerns) cannot readily be encapsulated with object-oriented units [39]. This mismatch results in *scattering*, i.e., parts of a requirements concern are distributed over many classes, and *tangling*, i.e., one class contains parts of many different requirements concerns. This problem is not applicable only to object-oriented systems but applies to any type of system description and has also been referred to as the tyranny of the dominant decomposition [141], as a chosen modularization technique inevitably will cause unwanted side-effects in the software design (e.g., objects cause scattering and tangling). A few examples for requirements concerns for which aspects provide a better encapsulation than objects are authorization/authentication, caching, concurrency management, debugging, distribution, logging, testing, transaction management, and even a feature or use case [61].

The term *concern* is rather vague and contentious. Generally, a concern encompasses anything related to a particular concept that is of interest to a stakeholder. In the area of requirements engineering, concerns are often non-functional requirements or qualities of a system, use cases or scenarios, entities in a domain model, stakeholders' high-level goals, or viewpoints. A defining characteristic of many of these concerns is that they *crosscut* other concerns multiple times (i.e., a concern that is scattered over other concerns is said to crosscut these concerns, and the properties of the concern must be merged with the other concerns). A crosscutting concern is termed an *aspect*. The main promise of aspect-oriented techniques is that they allow crosscutting concerns to be encapsulated, leading to better modularization and more maintainable, reusable, and scalable software systems. Furthermore, aspects allow technical dependencies to be aligned with logical, real-world dependencies. For example, an aspect-oriented Logger depends on domain entities to be logged, as logically should be the case. In object-oriented systems, however, a call dependency exists from a domain entity to the Logger, thus intro-

ducing an unnecessary dependency for purely technical reasons. In general, aspects require a new way of thinking about software systems much like object-orientation did many years ago.

Inspired by MDSOC principles, the general goal of Aspect-Oriented Software Development (AOSD) is to produce advanced separation of concerns mechanisms that support the modularization and management of important concerns regardless of whether they are crosscutting or not. Initially, aspect-orientation focused on the implementation level leading to an ever-growing number of tools that provide extensions for aspects to major programming languages [20]. Essentially, aspects identify *join points*, i.e., locations in a modeling or programming language, through *pointcuts*, i.e., parameterized expressions. All possible join points of a particular language or notation are defined in a *join point model* which must be established at the outset. Aspects also specify *advice* which is inserted at the identified join points by a process called *weaving*. A pointcut therefore defines the structural and behavioral context for the execution of advice. In AOP, advice is commonly inserted either *before*, *after*, or *around* (instead of) the join point based on a composition rule. The advice defines properties such as an aspect's own behavior or structure. The portion of the system that potentially can be changed by an aspect is referred to as the *base* relative to the aspect. As advice may change the behavior of already existing structural entities thus violating proper object-oriented modularization, entities external to an aspect are referenced in a structured way with the help of *intertype declarations*.

The terms defined in the above paragraph are AspectJ terms [24] but the concepts do apply to other flavors of AOP and Aspect-Oriented Modeling (AOM) in general. Drawing inspiration from AOP, AOM climbed the abstraction ladder to focus on requirements and design modeling techniques. In contrast to AOP, AOM uses terminology that is different to stress the wider range of techniques that AOM encompasses as compared to AOP. Pointcut expressions are often referred to as *patterns*, weaving is just one kind of aspect *composition*, and the more general term *properties* is used instead of advice. Through modularization of crosscutting concerns and abstraction from complicated crosscutting relationships such as scattering and tangling, AOM allows for *modular reasoning* about individual concerns and *compositional reasoning* about emergent system

properties. Applying concerns in a systematic and coherent fashion leads to a) an improved understanding of their interactions, inter-relationships, and conflicts, and b) helps identify trade-offs early on in the development life cycle. AOM approaches yield system descriptions that make it easier to directly link a conceptual concern to its associated problems and solutions, thus supporting reuse and simplifying the tracing of a concern across the software development lifecycle. Furthermore, AOM allows concerns to be added or removed by a simple plug-in mechanism for modules, facilitating the evaluation of different solutions to concerns.

2.3. Summary

This chapter gives an overview of the User Requirements Notation (URN), the only standard that currently integrates goal and scenario modeling in one framework. Similarly to many other modeling notations and languages, URN cannot effectively deal with cross-cutting concerns. Aspect-oriented techniques, however, help encapsulate such concerns. The following two chapters report on groundwork required for the extension of URN with aspect-oriented concepts. Chapter 3 unambiguously describes the semantics of URN while Chapter 4 assesses the extensions to URN proposed in Chapter 3. After these two chapters, Chapter 5 gives an intuitive, example-based introduction to the Aspect-oriented User Requirements Notation (AoURN).

Chapter 3. Use Case Maps 2.0

This chapter first motivates the need for the clarification of the semantics of the Use Case Maps (UCM) notation in the context of the development of the Aspect-oriented User Requirements Notation (AoURN) and explains the workflow pattern-based approach used for deciding which semantic interpretations are most adequate. Subsequently, UCM 2.0 including an updated URN metamodel is presented. The UCM notation is extended to ensure that there is only one interpretation for each modeling element, and to allow for a richer semantic interpretation of UCM models and for the resolution of longstanding issues with the UCM notation; i.e., issues with instances of UCM maps, plug-in bindings for components and responsibilities, as well as cancellation and exception behavior. This extension is the first major update of the UCM notation since its inception more than a decade ago. The update is relative to a baseline version of the URN standard (see Appendix A: URN Metamodel (Baseline Version)) that is the same as the latest version of the URN standard [60] but without any of the enhancement discussed here. While UCM 2.0 includes all extensions discussed here, the latest version of the URN standard [60] incorporates most but not all extensions proposed here.

3.1. Motivating the Need for Use Case Maps 2.0

More sophisticated modeling techniques for aspects in the requirements engineering world are based not only on syntactic model information but also on semantics [37][40][72][163]. As the AoURN technique intends to exploit semantic equivalences of URN models, the semantics of URN must be clearly defined. Fortunately, the Goal-oriented Requirement Notation (GRL) is defined precisely enough for the purposes of AoURN and has been stable for many years. The UCM notation has also changed very little since its beginnings in the early 1990s, but some UCM modeling elements such as constructs for alternatives, concurrency, and hierarchical structuring are interpreted in different ways. The introduction of a UCM traversal mechanism [64][66] capable of

highlighting a particular scenario in the UCM model and transforming the scenario into a message sequence chart (MSC) [58] represented the first step toward a standardized semantic interpretation of UCM models. The semantics of UCM has also been defined based on LOTOS [52], abstract state machines (ASM) [54][56], as well as Clocked Transition Systems (CTS) and Timed Automata (TA) [56]. While the interpretation of UCM based on these formalisms is compatible with the intuitive interpretation of the UCM notation by the traversal mechanism, they do not address longstanding issues with instances of UCM maps, plug-in bindings for components and responsibilities, as well as cancellation and exception behavior, and they do not support semantic variations of the UCM notation. Building on the existing traversal mechanism, the semantics of the UCM notation must be further clarified to avoid more misunderstandings. Consequently, the abstract and concrete syntax of UCM must also be extended to ensure that important semantic variations can be expressed with the notation. Use Case Maps 2.0 is the result of this clarification and extension. It is crucial for the success of UCM 2.0 that the focus of these clarifications and extensions is not just on providing a strong foundation for aspect-oriented UCM modeling but on a more holistic approach that ensures the continued suitability of UCM to current and new requirements engineering challenges.

In 2003, an in-depth and qualitative but informal comparison of UCM to other scenario-based requirements engineering notations discussed the advantages and disadvantages of the UCM notation [8]. As with any other language, UCM has to be re-evaluated from time to time in light of new technological development and new application domains such as business process modeling and analysis as well as web services. Many languages and techniques have been suggested for the description of workflows and business processes over the recent years. It is important for the future of UCM to understand the strengths and weaknesses of the UCM notation compared to these new languages because of the great similarity of UCM and these languages. At first glance, UCM is very well suited to describe workflow. As a general scenario notation, the structure and intent are similar to workflow languages and UCM has already been used for business process modeling [118][119][147]. Furthermore, more formal assessments exist for many workflow and business process languages [127][145][150][156][157][158], allowing for a comparison of these languages. Each assessment is based on a language's support for

generic workflow patterns [127][145][158]. The workflow patterns have been collected from workflow situations frequently encountered when modeling workflow, and assess a notation's applicability to workflow modeling but also general scenario modeling [127] (see the examples in this paper). To this effect, the original UCM notation is evaluated and improvements for UCM 2.0 suggested based on these generic workflow patterns [85][104]. Building on these suggestions, further issues with the UCM notation are addressed. The assessment is then extended to UCM 2.0, thus indicating to what extent UCM 2.0 is capable of describing commonly encountered scenarios and comparing the capabilities of UCM 2.0 to the main standards in the area of workflow and business process languages (i.e., the Business Process Modeling Notation (BPMN) [110], UML 2 Activity Diagrams [113], and the Business Process Execution Language for Web Services (BPEL4WS) [115]). The assessment also provides insight into how to evolve tool support and the traversal mechanism for UCM 2.0. Note that workflow languages may also be analyzed from data [125], resource [126], and exception handling [128] perspectives which are beyond the scope of this research.

3.2. Workflow Patterns

In 2000, an analysis of workflow languages resulted in the publication of 21 *workflow patterns* that describe typical control flow dependencies in workflow models [145]. In 2006, a new report [127] was published that revisits this initial set of workflow patterns, defining the workflow pattern more formally with Colored Petri-Nets and introducing further patterns to increase the total number of workflow patterns to 43. The patterns are divided into eight groups [158]. The first two groups address basic and advanced branching and synchronization patterns. The third discusses multiple instances of activities. The fourth deals with state-based patterns. The fifth covers cancellation and force completion patterns. The sixth, seventh, and eighth groups discuss iteration, termination, and trigger patterns, respectively. See Table 2 in Section 4.9 for a complete list of the workflow patterns and Sections 4.1 to 4.8 for an explanation of the individual workflow patterns.

Since the initial publication of these patterns, many workflow management systems (e.g., Staffware, WebSphere MQ Workflow, FLOWer, COSA, iPlanet, SAP Workflow, FileNet, jBPM, OpenWFE, and Enhydra Shark) and standards for business process

modeling and workflow modeling (e.g., BPEL, BPMN, XPDL, UML Activity Diagrams, and EPCs) have been assessed based on the collected workflow patterns [158]. Furthermore, a PhD thesis [71] was written “to establish a formal foundation for control-flow aspects of workflow specification languages, that assists in understanding fundamental properties of such languages, in particular their expressive power” [70]. The research led to the YAWL (Yet Another Workflow Language) initiative [159] with its goal to create a workflow language with direct support for all of the discovered workflow patterns. It is important to note that although the workflow patterns were initially observed for workflow systems, they have a much broader scope and can be used to evaluate general scenario description languages [127]. Workflow patterns are applicable regardless of whether interactions between the environment and a system or interactions between internal elements of a system are described.

3.3. Clarifying the Semantics of Use Case Maps

While the semantics of most path elements and structural elements of the UCM notation is clearly defined, there are a few that require further clarification to ensure consistent usage by all requirements engineers [66]. This section first lists the semantic variations of core UCM elements that have been observed over the years and defines each of the unclear UCM elements more precisely. The remainder of the section then introduces extensions to the UCM notation that enable requirements engineers to model the other semantic interpretations of the UCM elements in question.

1. Do OR-forks represent an exclusive OR or multiple choice? If OR-forks are multiple choice and more than one branch condition evaluates to true, are the branches executed in parallel? Are OR-forks without specified conditions allowed and if so, what does that mean?

The semantic interpretation of the core UCM path elements has to reflect the intuitive nature of the UCM notation. Therefore, an OR-fork must be interpreted as an exclusive OR, thus avoiding the case where alternative paths are potentially executed in parallel. OR-forks do not model multiple choice. OR-forks without specified conditions are possible, describing the case where a branch is not chosen explicitly by the branch condition but implicitly. The chosen branch is the one that contains the responsibility (or other path

element) that is started first. If at least one condition of an OR-fork is specified, then all other conditions of the OR-fork must also be specified.

2. Since dynamic stubs contain a selection policy that governs which plug-in maps are chosen, the same issues that apply to OR-forks also apply to dynamic stubs. In particular, what happens if more than one plug-in map is active? Are plug-in maps executed in parallel or in sequence? If plug-in maps are executed in parallel, is synchronization required before continuing past the dynamic stub?

In contrast to simple OR-forks, selection policies of dynamic stubs allow multiple choice. If more than one plug-in map is selected, then the plug-in maps are executed in parallel. The plug-in maps, however, are not synchronized before continuing past the dynamic stub. Similarly to OR-forks, dynamic stubs without a selection policy are possible and describe the case where a plug-in map is not chosen explicitly but implicitly. The chosen plug-in map is the one that contains the responsibility (or other path element) that is started first. If at least one plug-in map of a dynamic stub has a selection policy specified, then all other plug-in maps of the dynamic stub must also have a selection policy specified.

3. Do static stubs have a selection policy?

Conceptually, static stubs do not have a selection policy because the only plug-in map is always selected and the purpose of static stubs is simple hierarchical structuring. Technically in terms of the URN metamodel, however, all stubs have a selection policy, but the selection policy of a static stub is always true.

4. Are all outgoing branches of an AND-fork always executed? Do all incoming branches have to arrive at an AND-join before the scenario can continue?

Again, the semantic interpretation of the core UCM path elements has to reflect the intuitive nature of the UCM notation. Therefore, all branches of an AND-fork are activated in parallel and all incoming branches must arrive at an AND-join before continuing.

5. Are all start/end points of a map equivalent? Are all start points available for execution at all times? Are all start/end points available for continuation from/to parent maps?

There are cases where one would like to indicate that a start point on a map is available only once the map is active, i.e., a scenario described by the map is currently executing. This indication is, for example, useful to show that user input may occur only in the context of the map. In terms of end points, situations exist where it is desirable to indicate a local end to a scenario. Typically, such start/end points are not bound to in/out-paths of stubs, but remain strictly local. For UCM 2.0, one can hence distinguish regular and *local* start/end points. Figure 6 shows how local start/end points are indicated visually with a cross-out. The cross-out is chosen because such start and end points are not allowed to be selected for plug-in bindings.



Figure 6 Regular (left) and Local (right) Start/End Points

6. Do waiting places and timers remember arriving triggers or are these triggers transient? A transient trigger is forgotten if it arrives at a waiting place or timer and nothing is already waiting at the waiting place or timer to proceed.

Both interpretations are possible and the wait type attribute of waiting places and timers is used to distinguish between persistent and transient triggers. In the current UCM metamodel, the wait type is specified by a String variable, but for UCM 2.0 it should be formalized by an enumerated type. At this point, there is no graphical syntax for the wait type.

7. Do parallel branches in the same component imply interleaving or concurrency?

That depends on which kind of component is used. Interleaved routing is implied if the component of kind object is used, because the assumption is that a component of kind object can do only one thing at the time. A protected component, on the other hand, enforces the one-active-path-at-a-time rule. In this case, only one path is active and further paths have to wait for its completion (i.e., the execution order is neither interleaving nor concurrent but sequential). For other component kinds, concurrent execution can be assumed.

8. What are the semantics of dynamic responsibilities, pools, and dynamic components?

These concepts are no longer supported in UCM 2.0 because of their unclear semantics and because they a) were initially meant for low-level object-oriented operations related to references and visibility, b) are not really used in practice, and c) are now covered largely by component types as well as component and responsibility plug-in bindings introduced in Section 3.6.

The latest version of the URN standard [60] has addressed all of the above clarifications except for OR-forks without specified conditions, dynamic stubs without a selection policy, and local start/end points which are all not supported.

Modifications of the Traversal Mechanism and the URN Metamodel – OR-forks without specified conditions and dynamic stubs without a selection policy require changes to the specification of scenario definitions and the traversal mechanism as it needs to understand the updated scenario definition. The URN metamodel of the updated scenario definitions (Figure 7) contains the new class `DeferredChoiceInitialization` and a composition with `ScenarioDef`. `DeferredChoiceInitialization` describes where the deferredChoice occurs and which `enabledBranch` or `enabledPlugin` to follow after the deferredChoice. The deferredChoice is either an OR-fork in which case the `enabledBranch` determines which out-going branch of the OR-fork to choose or the deferredChoice is a dynamic stub in which case the `enabledPlugin` determines which plug- of the dynamic stub to choose.

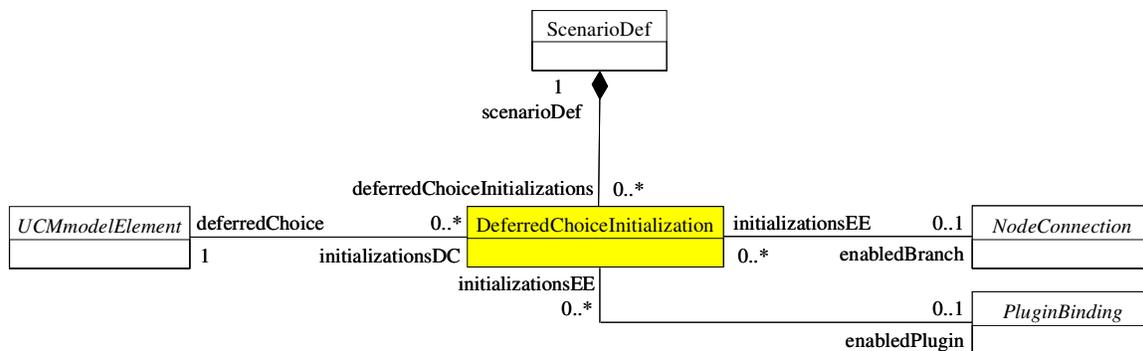


Figure 7 URN Metamodel Extensions: Deferred Choice

The traversal mechanism also needs to understand the updated start point, end point, and waiting place concepts. `StartPoints` and `EndPoints` have the new Boolean attribute `local` (Figure 8). It differentiates local (true) and regular (false) start and end points. The `waitType` attribute of a `WaitingPlace` (and therefore also of a `Timer`) is now a `WaitKind` enu-

meration, indicating whether the waiting place or timer has Transient or Persistent characteristics (Figure 8).

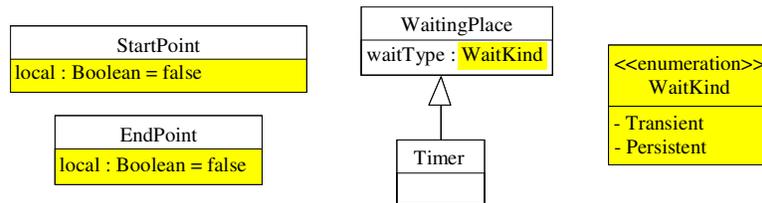


Figure 8 URN Metamodel Extensions: local and waitType

3.4. Synchronizing Stub

Considering the above clarifications, the following can still not be modeled concisely with the core elements of the UCM notation:

- Synchronization of plug-in maps of dynamic stubs (i.e., parallel, multiple choice paths),
- Activation of only a subset of all specified parallel branches, and
- Continuation of a scenario if only a subset of all specified incoming branches has arrived at a synchronization point.

The *synchronizing stub* is a new UCM 2.0 path element that is designed to address these issues. The synchronizing stub is a dynamic stub which allows its plug-in maps to be executed in parallel and continues only once the plug-in maps have synchronized. It is different to an AND-fork / AND-join combination where the branches of the AND-fork represent the plug-in maps of the synchronizing stub. The branches of an AND-fork are always executed whereas this is not the case for synchronizing stubs. The selection policy of the synchronizing stub governs which plug-in maps are to be executed. Similarly, an AND-join expects all of its incoming branches to arrive whereas this is not the case for synchronizing stubs.

By default, a synchronizing stub expects as many plug-in maps as chosen by its selection policy to arrive at an out-path before the scenario is continued (i.e., not necessarily all plug-in maps defined for the stub). A *synchronization threshold* may override the default. It is an integer expression greater than zero and may be defined for each out-path of a stub. The synchronization threshold may be greater than the number of plug-in maps defined for the stub, because a single plug-in map may arrive at a stub's out-path

multiple times due to loops. All plug-in maps are ignored when they arrive at a stub's out-path after its synchronization threshold is reached, i.e., the path does not continue.

Figure 9 gives an example for a synchronizing stub, identified by the S inside the stub's diamond symbol. Both of its out-paths have synchronization thresholds defined in square brackets (3 and 2). The superscript X inside the stub indicates that a replication factor is defined for at least one plug-in map. A *replication factor* specifies how many instances of a plug-in map are to be executed in parallel. The replication factors for each plug-in map are shown in parentheses whereas the selection policy is shown in brackets for each plug-in map. For example if condC1 is true, one instance of map P1 and three instances of map P2 are executed in parallel. If map P1 terminates at the fail end point and all instances of map P2 at the success end point, the path on map M will continue along the success path, because the three instances of map P2 arriving at that out-path satisfy the synchronization threshold. The fail path on map M is not yet executed because only one plug-in map instance arrived (i.e., map P1) instead of the required two. In a situation where both synchronization thresholds are satisfied, the success and the fail paths continue in parallel.

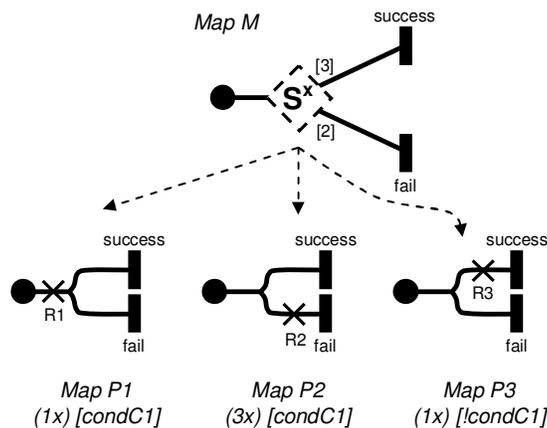


Figure 9 Synchronizing Stub

A synchronizing stub may further be characterized as blocking or cancelling. *Blocking* means that the stub cannot be entered a second time along the same in-path before all initiated plug-in maps have completed, while *cancelling* means that remaining plug-in maps are aborted, if the required number of plug-in maps have arrived at each out-path. Note

that cancelling synchronizing stubs are not supported by the latest version of the URN standard while blocking synchronizing stubs are supported.

Modifications of the Traversal Mechanism – The traversal mechanism needs to be enhanced to understand synchronizing stubs. For the default behavior, the mechanism needs to execute plug-in maps in parallel, keep track of the required number of plug-in maps that have to complete before path traversal is allowed to continue past the stub, and track the number of completed plug-in maps per out-path. In the default case, each out-path is followed if it is triggered as often as plug-in maps are started in parallel. If a synchronization threshold is specified, the traversal mechanism must only follow an out-path once the required number of triggers have arrived. If a replication factor is defined for a plug-in map, the traversal mechanism must also create multiple instances of the plug-in map that are executed in parallel.

Modifications of the URN Metamodel – Minor updates to the URN metamodel are required for the synchronizing stub. The attributes synchronizing, blocking, and cancelling need to be added to Stub, the attribute replicationFactor needs to be added to PluginBinding, and the attribute threshold needs to be added to NodeConnection (Figure 10).

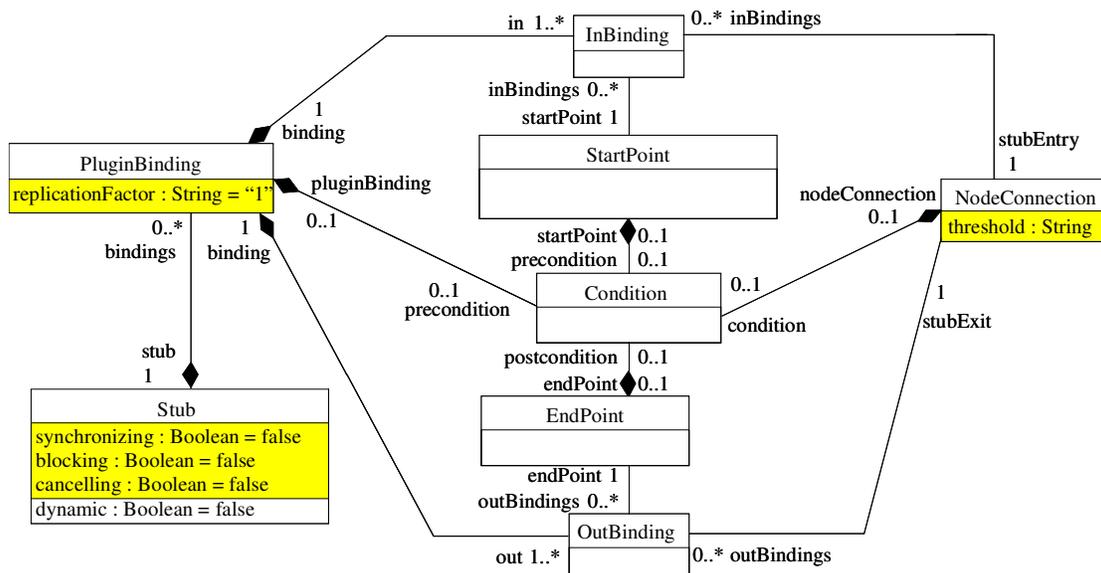


Figure 10 URN Metamodel Extensions: Synchronizing Stub

3.5. Instances of UCM Maps

This section discusses a semantic limitation that relates to the current usage of UCM plug-in maps in multiple situations, and thus goes beyond the semantic interpretation of individual UCM path and structural elements. Note that the latest version of the URN standard has addressed this limitation. The current traversal/execution mechanism for UCM assumes that each map exists only once at runtime in the whole system described by the UCM model. Therefore, all stubs that have a particular map as a plug-in map in the UCM model share that same map at runtime. Currently, a stub cannot use its own private instance of a plug-in map at runtime. If one stub or a group of stubs needs to use a separate instance of the map at runtime, the map must be duplicated in the UCM model. This duplication occurs quite frequently when reusing a map that describes a pattern or when the map contains a synchronization point which must not be shared by everyone. This rather arbitrary semantic limitation is obviously problematic from a maintenance point of view and defeats one major purpose of plug-in maps – reuse. There should always be only one instance of a map in the UCM model, but possibly many instances of the map at runtime.

3.5.1 Singleton Maps

In general, there are three cases that should be modeled easily.

1. All stubs share the same map at runtime. Therefore, the map is a singleton (i.e., only one instance of the map exists in the whole system).
2. Each stub uses a different instance of the map at runtime.
3. Each different group of stubs uses a different instance of the map at runtime.

For UCM 2.0, one can hence distinguish between a *singleton map*, i.e., a map for which only one instance exists at runtime in the system described by the UCM model, and a map that can be instantiated multiple times at runtime. Therefore, a map does not need to be repeated in the UCM model, i.e., there is always only one instance of the map in the UCM model, while there may be many instances of the map at runtime. Hence, an instance is referring to a runtime concept and not a model concept in the remainder of this section.

The fact that stubs are often used to restructure a complicated UCM map (e.g., map M in Figure 11) into two or more smaller maps (e.g., map C with a stub to which plug-in map D is bound) has implications for instances of UCM maps. M and the two maps C and D must remain equivalent. To guarantee this equivalence, the stub on C must contain exactly one instance of the plug-in map D at any time. Otherwise, flattening C with more than one plug-in map instance would not generate the original map M. In Figure 11, M is equivalent to C and D, but is not equivalent to C and the two instances D1 and D2. Only if a second instance of the stub on C exists, may a second instance of the plug-in map D be created. This situation may occur if a second instance of C exists, possibly because two maps use C (see A and B in Figure 11), which also means that another instance of the original map M exists. Consequently, each instance of a stub always contains exactly one instance of a plug-in map at any time.

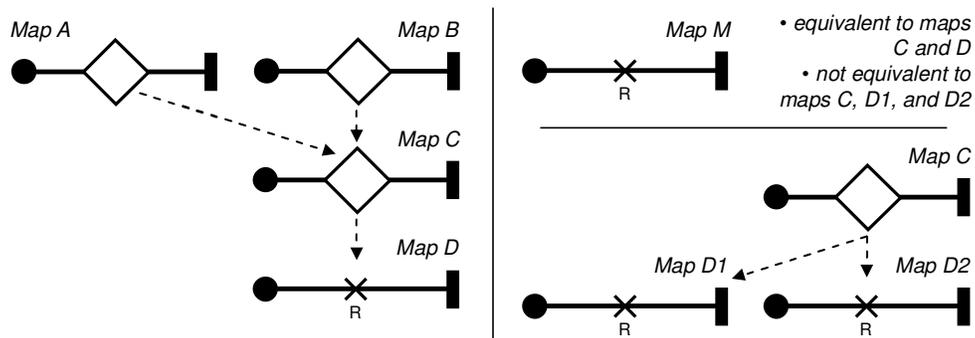


Figure 11 Singletons and Map Instances

Note that this does not mean that a different stub always has to use a different plug-in map instance. In Figure 11 if C is a singleton, then A and B will use the same instance of C. Since only one instance of C exists, only one instance of the stub on C exists, and therefore only one instance of D. If, however, C is not a singleton, then A and B will use different instances of C. Since two instances of C exist, two instances of D will also be created, unless of course D is a singleton, in which case only one instance of D exists regardless of C being a singleton or not.

Consideration must also be given to the situation where a plug-in map is used multiple times for one stub (e.g., if the stub to which it is bound is in a loop). If an in-path of the stub is traversed for a second time, either a new instance of the plug-in map could be created or the traversal mechanism could continue using the already existing instance

of the plug-in map. For similar reasons as discussed in the previous paragraph, the existing instance of the plug-in map must be used. The same applies to stubs with multiple in-paths that may be traversed at different times. The same instance of the plug-in map is always used for the stub.

Given the ability to designate a map as a singleton, Cases 1 and 2 listed at the beginning of this section are easily modeled. Figure 12 shows how the third case can be modeled. As indicated, maps G and H are singletons. Therefore, the group of stubs in maps A, B, and C uses one instance of map I (the one bound to the stub in G). The group of stubs in maps D, E, and F uses another instance of I (the one bound to the stub in H). Therefore, an intermediary layer of singleton maps allows Case 3 to be modeled straightforwardly.

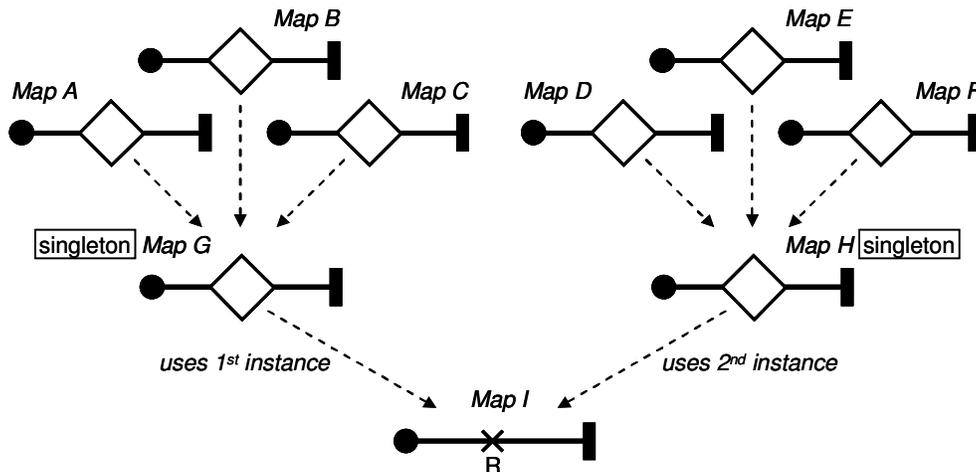


Figure 12 One Instance for Each Group of Stubs

The semantics for dynamic stubs is similar to static stubs in that a dynamic stub may contain only one instance of each of its plug-in maps at a time. The semantics differs of course as the purpose of a dynamic stub is to model AND-forks and OR-joins in addition to simple hierarchical structuring (see left side of Figure 14). Each in-path is equivalent to an AND-fork that is connected to the flattened plug-in maps according to the specified plug-in bindings. Analogously, each out-path corresponds to an OR-join connected to the flattened plug-in maps based on the specified plug-in bindings.

The semantics of an AND-fork in a flattened UCM model corresponds to the semantics of the selection policy of stubs and not the semantics of regular AND-forks in

non-flattened models, i.e., it allows multiple choice which can be modeled by guards on the branches of the AND-fork. The URN metamodel already allows for these guards even though they are not used in standard URN models.

Modifications of the Traversal Mechanism – To support instances of maps, the traversal mechanism has to keep track of the number of instances created per non-singleton map and which stub instance is using which plug-in map instance. When traversing the model, each parallel branch must know which instance of a map it is traversing. Furthermore, the traversal mechanism must allow synchronization of parallel branches only if the branches traverse the same map instance.

Modifications of the URN Metamodel – The changes to the URN metamodel required for support of map instances are minimal. Only one attribute indicating whether a map is a singleton must be added to UCMmap (Figure 13).

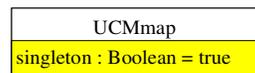


Figure 13 URN Metamodel Extensions: Instances of UCM Maps

3.5.2 Synchronizing Stubs and Instances of UCM Maps

The semantics for synchronizing stubs in terms of instances of maps, however, is slightly different than for static and dynamic stubs. The plug-in maps bound to a synchronizing stub have to act as one group because they belong to the same AND-fork / AND-join pair represented by the synchronizing stub. Only plug-in maps in the same group must be allowed to synchronize with each other. Furthermore, the blocking and cancelling attributes of synchronizing stubs operate on the group of plug-in maps and rely on these groups not being mixed together. Therefore, if an in-path of the synchronizing stub is visited for a second time, a second group of plug-in maps must be created. Consequently, a synchronizing stub may contain more than one instance of a plug-in map at the same time. This required group behavior leads to the notion of *visit*. A visit is related to the number of times an in-path is visited during a scenario. For each visit, a distinct set of plug-in map instances exist for the synchronizing stub. Since a stub may have multiple in-paths that may be traversed at different times during a scenario, the number of visits may not be the

same for each in-path. Hence, multiple sets of plug-in map instances may be active for a synchronizing stub.

This behavior, however, is equivalent to having one instance for each plug-in map with multiple scenarios executing between AND-forks and AND-joins that can synchronize only if they passed an AND-fork during the same visit. One can think of a scenario carrying the visit information from the AND-fork and the AND-join ensuring that only scenarios with the same visit number synchronize. The synchronizing stub is therefore still conceptually equivalent to its flattened counterpart (see right side of Figure 14) with each in-path converted to an AND-fork and each out-path converted to an AND-join. The connections of the AND-fork and AND-join to the flattened plug-in maps are again based on the specified plug-in bindings.

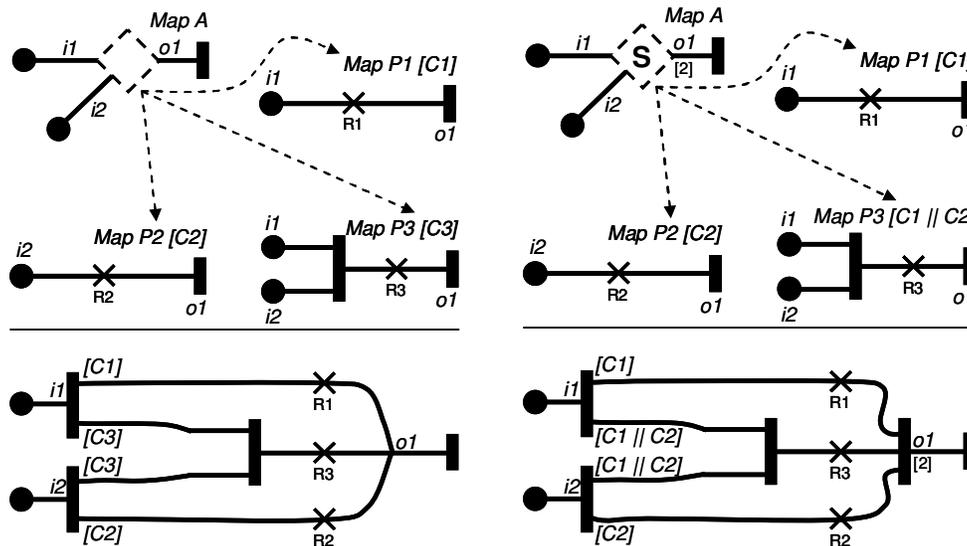


Figure 14 Flattened Dynamic and Synchronizing Stubs

The semantics of AND-forks in flattened UCM models is the same as for dynamic stubs explained earlier. The semantics of AND-joins corresponds to synchronizing stubs and not the regular AND-joins in non-flattened models. Thus, they allow the specification of synchronization thresholds. Again, the URN metamodel already allows for these thresholds even though they are not used in standard URN models and a constraint for the use of thresholds must therefore be relaxed for flattened UCM models.

If a synchronizing stub has only one in-path, then the creation of multiple plug-in map instances is straightforward. The creation of instances becomes more complex when multiple in-paths are present as illustrated in Table 1 and Figure 15.

Table 1 Instances and Synchronizing Stubs

#	In-path	Resulting Action {specified synchronization threshold}	Resulting Action {default synchronization threshold}
1	i1	create 1 st P1 and 1 st P2; continue with i1 on 1 st P1	create 1 st P1 and 1 st P2; set synchronization threshold to 2; continue with i1 on 1 st P1
2	i2	continue with i2 on 1 st P1 (for the second time on that instance) and 1 st P2	continue with i2 on 1 st P1 (for the second time on that instance) and 1 st P2 {the synchronization threshold is reached and traversal continues past the stub}
3	i3	continue with i3 on 1 st P2 {the synchronization threshold is reached and traversal continues past the stub}	continue with i3 on 1 st P2 {ignore arrival at out-path}
4	i1	create 2 nd P1 and 2 nd P2; continue with i1 on 2 nd P1	create 2 nd P1 and 2 nd P2; set synchronization threshold to 2; continue with i1 on 2 nd P1
5	i1	create 3 rd P1 and 3 rd P2; continue with i1 on 3 rd P1	create 3 rd P1 and 3 rd P2; set synchronization threshold to 2; continue with i1 on 3 rd P1
6	i2	continue with i2 on 2 nd P1 (for the second time on that instance) and 2 nd P2	continue with i2 on 2 nd P1 (for the second time on that instance) and 2 nd P2 {the synchronization threshold is reached and traversal continues past the stub}
7	i3	continue with i3 on 2 nd P2 {the synchronization threshold is reached and traversal continues past the stub}	continue with i3 on 2 nd P2 {ignore arrival at out-path}
8	i3	continue with i3 on 3 rd P2	continue with i3 on 3 rd P2
9	i3	create 4 th P1 and 4 th P2; continue with i3 on 4 th P2	create 4 th P1 and 4 th P2; set synchronization threshold to 2; continue with i3 on 4 th P2

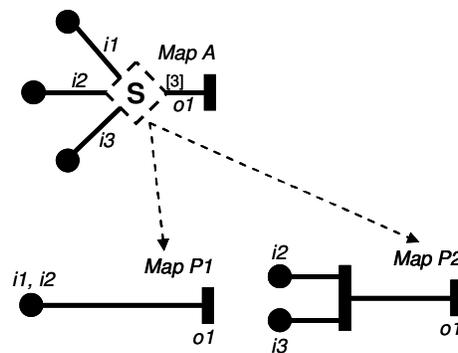


Figure 15 Instances and Synchronizing Stubs

If an in-path is visited for the first time after a different in-path was visited the first time, then no new plug-in map instances need to be created, because both traversals belong to the same visit. See Figure 15 and the first, second, and third columns of Table 1 for an example. Given a synchronizing stub with three in-paths i_1 , i_2 , and i_3 , two plug-in maps P1 and P2 bound to the stub as indicated in Figure 15, and a traversal order of the in-paths (i_1 : 1st, 4th, 5th; i_2 : 2nd, 6th; i_3 : 3rd, 7th, 8th, 9th), there will be four visits where instances of both plug-in maps P1 and P2 are created. Instances are created in groups, because all plug-in maps of the synchronizing stub have to act as a group. The first instances of P1 and P2 are created at the first traversal of i_1 . The second and third traversals do not cause new instances to be created because these in-paths have not yet been used for the first instances, and hence are part of the same visit. The fourth traversal creates the second set of instances (second visit) because i_1 is traversed for the second time. The fifth traversal creates the third set of instances because i_1 is again traversed. The sixth and seventh traversals use the instances of the plug-in maps that belong to the second visit because the events go to the longest-waiting instance of a plug-in map. The eighth traversal uses the third set of instances. Finally, the ninth traversal causes the fourth set of instances to be created because i_3 is traversed for the fourth time. Note that the traversal of in-paths is important for the creation of plug-in map instances but not the traversal of start points on the plug-in map (e.g., P1's start point is traversed five times because i_1 and i_2 are both bound to the start point). Furthermore, if a replication factor is defined for a plug-in map, then not one instance of the plug-in map is created each time but as many as specified by the replication factor.

If the synchronization threshold is not specified in the example in Figure 15, then the default behavior stipulates that as many plug-in map instances must arrive at the out-path as are executed in parallel before the traversal is allowed to continue. The fourth column in Table 1 explains the behavior of the synchronizing stub in this case assuming that the selection policy selects both plug-in maps. Note that the synchronization threshold is always specified upon first arrival at a stub during each visit. Subsequent arrivals during the same visit along other in-paths do not change the synchronization threshold for that visit, even if the number of plug-in map instances that are being traversed changes.

In summary, the following rules have to be followed for instances of UCM maps. If a map is a singleton, it exists only once at runtime in the whole system described by the UCM model. If it is not, then several instances of the map may be created at runtime when traversing the model. An instance may be created only when the traversal reaches a stub to which the map is bound. An instance of the map is created upon first use of the stub. No other instance must be created for the stub (i.e., the same instance is used for this instance of the stub throughout the life of the stub's map; a different instance of a plug-in map, however, may be used for a different instance of the stub). If the stub is a synchronizing stub, the last rule is weakened to allow new plug-in map instances to be created for each visit of the same synchronizing stub as explained earlier. Consequently, a plug-in map of a synchronizing stub is constrained to a non-singleton map in the UCM model, since it is possible for multiple instances of a plug-in map of a synchronizing stub to exist at runtime.

Modifications of the Traversal Mechanism – To support synchronizing stubs and instances of maps, the traversal mechanism has to keep track of how often an in-path/out-path of the stub has been traversed to determine the visit. When traversing the model, each parallel branch must know to which visit it belongs. Furthermore, the traversal mechanism must allow synchronization of parallel branches only if the branches belong to the same visit.

3.6. Plug-in Bindings for UCM Components and Responsibilities

3.6.1 Component Plug-in Bindings

This section first discusses a semantic ambiguity concerning the relationship of components on parent maps and plug-in maps, again going beyond the semantic interpretation of individual UCM path and structural elements. Note that the latest version of the URN standard has addressed this ambiguity. The continuation of a path from a stub to its plug-in maps is well defined with the help of plug-in bindings that connect the stub's in-paths with start points on the plug-in maps and the end points on the plug-in maps with the stub's out-paths. The relationship of components on a parent map and components on a

plug-in map, however, is not that clear at all. The simple example in Figure 16 allows for three different interpretations.

1. C2 is contained in C1 because the stub is inside C1 (i.e., the stub is used for a refinement of the scenario).
2. C1 and C2 are the same component because C2 describes a role in a pattern that is reused with the help of the stub (i.e., C1 is playing the role described by C2).
3. C2 provides a service that is being used by C1. Therefore, C1 and C2 are different components, and C2 is not contained in C1.

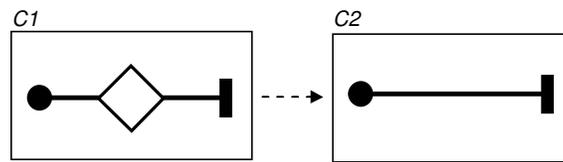


Figure 16 Components on Parent (left) and Plug-in (right) Maps

These three cases could be differentiated with a naming scheme for plug-in maps. However, if the stub is not bound to a component or when multiple components are used on the parent map and the plug-in map, the ambiguities cannot easily be resolved through a naming scheme alone. Furthermore, a plug-in map may not contain a component at all. Does this mean that the path on the plug-in map resides within the component of the stub? Alternatively, should the path be considered unbound? To clearly specify the relationship of components on the parent map with components on the plug-in maps, the current plug-in bindings are extended from UCM paths to UCM components in UCM 2.0.

The UCM model in Figure 17 contains one map with a stub and four plug-in maps. *Component plug-in bindings* connect component C1 to the component with the keyword parent on the plug-in maps. This approach allows plug-in map b) to model Case 1 from the list at the beginning of this section, plug-in map c) to model Case 2, and plug-in map d) to model Case 3. Plug-in map a) shows the case where the path on the plug-in map is not bound to any component (not even component C1). Consequently, the location of a stub relative to components on the parent map does not have any semantic significance.

With the help of plug-in bindings for components, the relationship of multiple components on the parent map and on plug-in maps can also be modeled easily. Components on plug-in maps for which plug-in bindings are supposed to exist are identified by the keyword *parent*, a prefix for the component name as shown in Figure 17 and Figure 18.

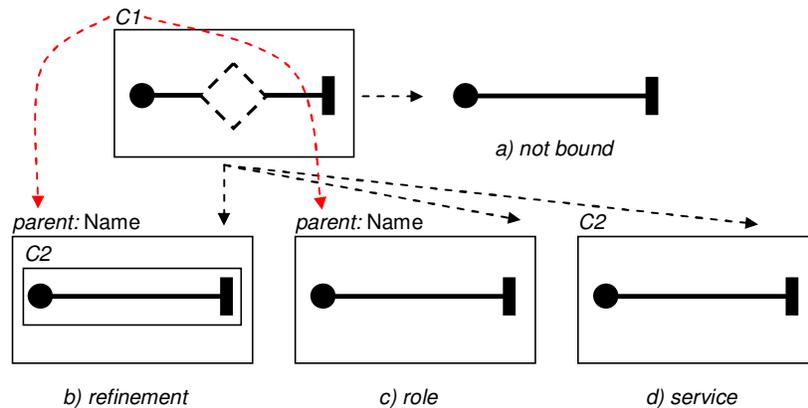


Figure 17 Plug-in Bindings for Components

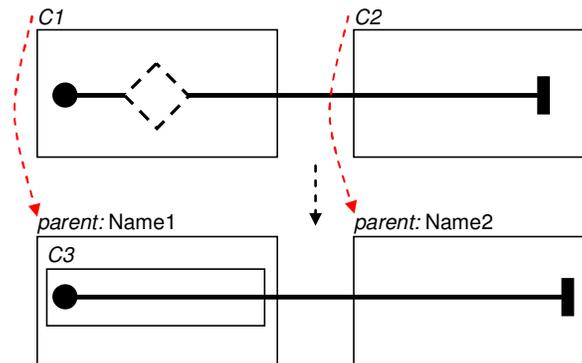


Figure 18 Plug-in Bindings for Multiple Components

Modifications of the Traversal Mechanism – To support plug-in bindings for components, the traversal mechanism now has to substitute a component on the plug-in map with the connected component from the parent map if a component plug-in binding is specified.

Modifications of the URN Metamodel – The following changes to the URN metamodel are required for support of plug-in bindings for components (Figure 19). A new composition needs to be added between ComponentBinding and PluginBinding. ComponentBinding is a new element that defines which components are connected with

the help of two associations to ComponentRef called parentComponent and pluginComponent. A new attribute for Component (context) indicates whether a component on a plug-in map is supposed to be used in a component plug-in binding, so that the component name can be prefixed with the keyword *parent* on the map.

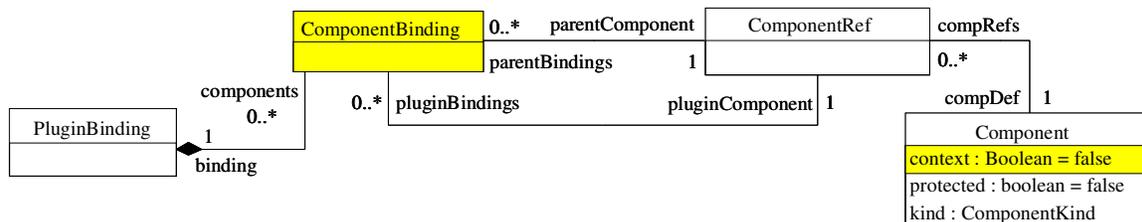


Figure 19 URN Metamodel Extensions: Component Plug-in Bindings

3.6.2 Responsibility Plug-in Bindings

The second change to plug-in bindings relates to responsibilities. Often, plug-in maps describe patterns which may be used in different contexts. As such, a tailoring of the pattern to a particular context may be required. Component plug-in bindings allow components on the plug-in map to be replaced according to the instructions from the parent map. Similarly, *responsibility plug-in bindings* allow a responsibility on the plug-in map to be replaced with another responsibility from the model. The instructions are again specified by the parent map and a responsibility for which a responsibility plug-in binding is supposed to exist is indicated again with the keyword *parent*.

Component and responsibility plug-in bindings combined enable, e.g., the abstract modeling of a security service with a Security Provider component and an authenticate responsibility on a plug-in map. This abstract plug-in map may then be tailored with a concrete Access Card Reader component and a concrete swipeCard responsibility. The example in Figure 20 shows that the stub to the left replaces responsibility R1 with R2 while the stub on the right replaces R1 with R3.

Modifications of the Traversal Mechanism – To support plug-in bindings for responsibilities, the traversal mechanism now has to substitute a responsibility on the plug-in map with the connected responsibility from the model if a responsibility plug-in binding is specified.

Modifications of the URN Metamodel – The following changes to the URN metamodel are required for support of plug-in bindings for responsibilities (Figure 21). A

new composition needs to be added between ResponsibilityBinding and PluginBinding. ResponsibilityBinding is a new element that defines which responsibilities are connected with the help of an association to RespRef called pluginResp (the responsibility on the plug-in map) and an association to Responsibility called parentResp (the responsibility from the model). A new attribute for Responsibility (context) indicates whether a responsibility on a plug-in map is supposed to be used in a responsibility plug-in binding, so that the responsibility name can be prefixed with the keyword *parent* on the map.

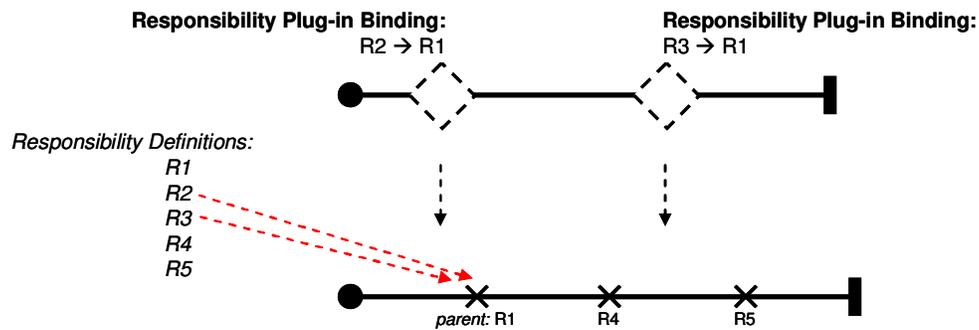


Figure 20 Plug-in Bindings for Responsibilities

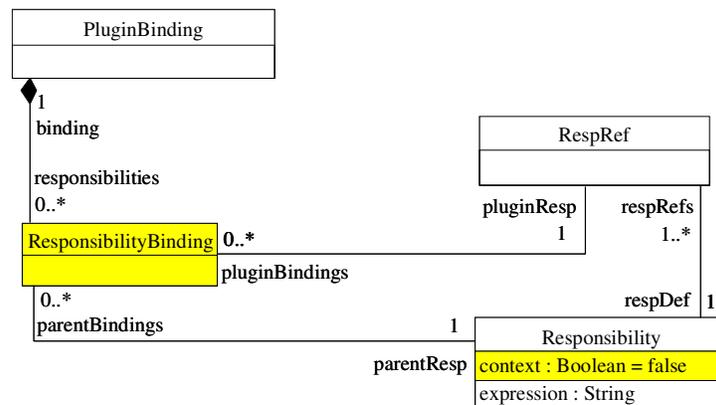


Figure 21 URN Metamodel Extensions: Responsibility Plug-in Bindings

3.7. Modeling Cancellations and Exceptions in Use Case Maps

While synchronizing stubs cover some cancellation situations, general support for cancellations and exceptions is not provided by the UCM notation. Note that the latest version of the URN standard also has not addressed cancellation issues. In the original UCM language, path elements for failure points and aborts existed but have not seen widespread use. The current jUCMNav tool does not support either concept. With failure points, ex-

ceptions are raised explicitly at a specific point on a path, thus cancelling the rest of the path. The recovery behavior required for a failure point, however, has to be modeled with rather complex maps involving concurrent paths, loops, and timeouts (Figure 22).

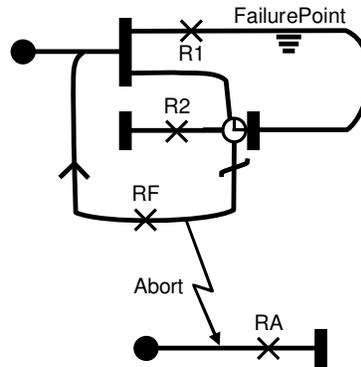


Figure 22 Original Failure Points and Aborts

Furthermore, the impact of a failure point on the UCM model is limited to cancelling the remaining portion of the path but does not address other concurrent paths. Aborts, on the other hand, explicitly indicate the cancellation of another path. Figure 22 shows the abort as a lightning bolt drawn from the path that raises the abort to the other path that is cancelled because of it. The explicit way of the original UCM language for specifying failure points and aborts makes it a) impractical to model aborts across different maps, and b) difficult for failure points and aborts to model cancellations and exceptions that may occur at any point in a scenario.

The cancellation and exception framework of UCM 2.0 addresses these issues. It still supports an explicit way of modeling failure points but also supports an implicit way of modeling failure points that makes it easier to model cancellations and exceptions that may occur at any point in a scenario. UCM 2.0 also simplifies the specification of aborts to allow aborts across map boundaries to be defined. Generally, UCM 2.0 failure points in conjunction with UCM 2.0 abort mechanisms operate much like exceptions in object-oriented programming languages such as Java.

3.7.1 Explicit Failure Points and Abort Mechanisms

The *explicit* approach draws on the original *failure points* to indicate explicitly the location of a failure on a path. However, each failure point also defines more formally a *fail-*

ure condition specifying when a failure occurs and a *failure expression* which is used to indicate which failure has occurred. A failure point is therefore a new choice point in the UCM model.

When the failure point is reached during a scenario, the failure condition is evaluated. If the failure condition of a failure point evaluates to false, path traversal continues normally past the failure point. However, if the failure condition evaluates to true, then the remaining portion of the path is cancelled, and the failure expression indicates which failure occurred by setting a *failure variable* to true. Essentially, the failure point is raising an exception which is the name of the failure variable.

The recovery behavior in response to a failure is now modeled in a much more structured way with the help of failure and abort paths, replacing the original abort construct. Failure and abort paths respond to failures specified by the failure points. Two new kinds of start points – the *failure start point* and the *abort start point* – indicate these two types of paths. Failure and abort start points have *guarding conditions* that indicate to which failure to respond.

When a failure occurs, the guarding conditions of all failure and abort start points in the UCM model are evaluated. The guarding condition of a failure or abort start point uses the variables set by the expressions of failure points. A failure or abort start point may respond to many failures. In this case, the guarding condition is a disjunction of variables set by failure points. In any case, those failure and abort start points whose guarding conditions evaluate to true are chosen to continue the path traversal in parallel. When continuing along a failure or abort path, each failure variable set by a failure point is reset to false if it is used in the disjunction of the failure or abort start point and if it evaluates to true.

A failure start point is indicated by the F inside the start point, while an abort start point is indicated by the F inside and the lightning bolt next to the start point. The example in Figure 23 shows the same scenario as in Figure 22 except for the abort behavior. The failure occurs if variable x is greater than three (as specified by the failure condition). The FailurePoint's failure expression (*failure = true*; is not shown visually) then sets the variable *failure* to true and the same variable is used by the failure start point as its guard-

ing condition. This technique allows the definition of failure handling behavior that covers large portions of a UCM model.

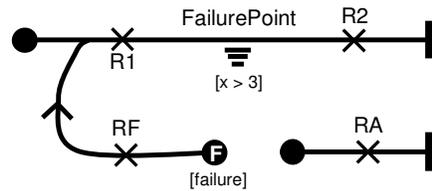


Figure 23 Failure Point and Failure Path

The difference between a failure start point and an abort start point is that the abort start point stops the traversal of any other concurrent branch in the *abort scope* in addition to the path where the failure occurred. The scoping mechanism is tied to the hierarchy of maps as established by the traversal mechanism. Only concurrent branches that are active on the same or lower level maps are cancelled. Note, however, that paths on lower level maps that are not connected to the map with the abort start point are not aborted.

The example in Figure 24 is now semantically equivalent to the scenario in Figure 22 because the abort start point aborts the path with responsibility RA. In general, the maps describe two situations: either the failure occurs or it does not occur. In the case where the failure does not occur, both maps start at their start points at the top left. In Figure 22, the scenario then continues along both branches of its AND-fork, reaches R1, passes through the failure point, and synchronizes at the timer. Hence, the scenario continues with the path with R2 and ends. In Figure 24, the scenario reaches R1, passes through the failure point, reaches R2, and ends. In the case where the failure does occur, again both maps start at their start points at the top left. In Figure 22, the scenario continues along both branches of its AND-fork, reaches R1, and then stops at the failure point. Consequently, the two concurrent branches do not synchronize at the timer and the timeout path is taken. On the timeout path, the abort of the path with RA occurs and then RF is reached before the whole scenario is started over. In Figure 24, the scenario reaches R1 and then stops at the failure point. Consequently, the scenario continues at the abort start point, the path with RA is aborted and RF is reached before the whole scenario is started over.

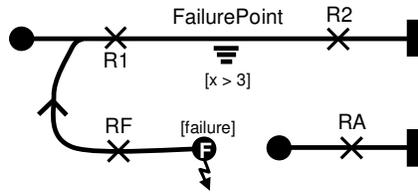


Figure 24 Failure Point and Abort Path

The example in Figure 25 further illustrates the abort scope. The hierarchy of the maps is given by the plug-in structure. Map B contains two disjoint paths. The top path is bound to the stub on map A, while the bottom path is not bound at all. The number of branches that are aborted if the failure occurs depends on which map specifies the abort path. If the abort path is specified on map A, then all paths on all maps are aborted except for the bottom path on map B, because no plug-in binding exists between the stub on map A and the bottom path on map B. The bottom path is not aborted because it cannot be reached from map A. If specified on map B, then only the two disjoint paths including both concurrent paths on B are aborted. If specified on map C, then all paths on C and map D are aborted. Finally, if specified on D, then only the path on D is aborted.

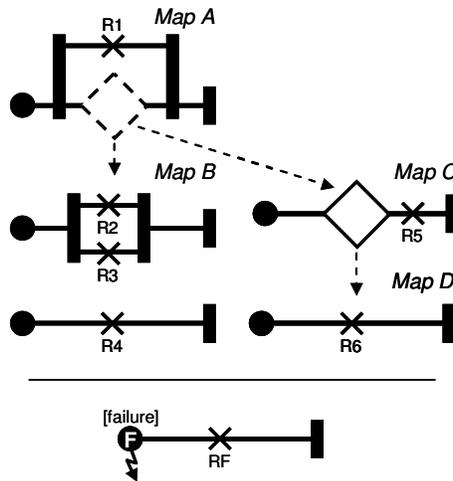


Figure 25 Scoping Mechanism for Abort Start Points

If the failure occurs in the example in Figure 25, the scenario ends with the end point of the abort path. It may be desirable to continue with the scenario once the failure is dealt with. This continuation is easily achieved with UCM 2.0 since the end points of a failure or abort path may be merged with other paths (Figure 23 and Figure 24) or bound to stubs just like any other end point. For example, if the failure point and abort path are specified

on map B and the end point of the abort path is bound to the out-path of the dynamic stub in map A, then the scenario will be able to continue on map A after the failure has occurred. Failure or abort start points, however, cannot be bound to in-paths of stubs. It is also not possible to connect end points or asynchronous paths to failure and abort start points.

Modifications of the Traversal Mechanism – The traversal mechanism needs to evaluate failure conditions, set failure variables, enable failure and abort start points based on the evaluation of guarding conditions and scenario definitions, and reset failure variables once the failure and abort paths are chosen. Finally, the traversal mechanism must be able to cancel concurrent branches in the scope of an abort path.

Modifications of the URN Metamodel – The changes to the URN metamodel, however, are rather small (Figure 26). A new path node FailurePoint needs to be added that is connected to exactly one previous and one next path node. A FailurePoint has an expression attribute that specifies the failure variable. Furthermore, the new attribute kind of type FailureKind must be added to StartPoint. The enumeration FailureKind can have the values None, Failure, or Abort. Note that it is already possible to define failure conditions for FailurePoints with the help of the existing condition for NodeConnections and guarding conditions for failure or abort start points with the help of the existing precondition for StartPoints. Also note that the existing condition for a NodeConnection expresses the condition to continue along the NodeConnection, i.e., beyond the failure point. Therefore, the negation of the failure condition is stored in the condition for a NodeConnection.

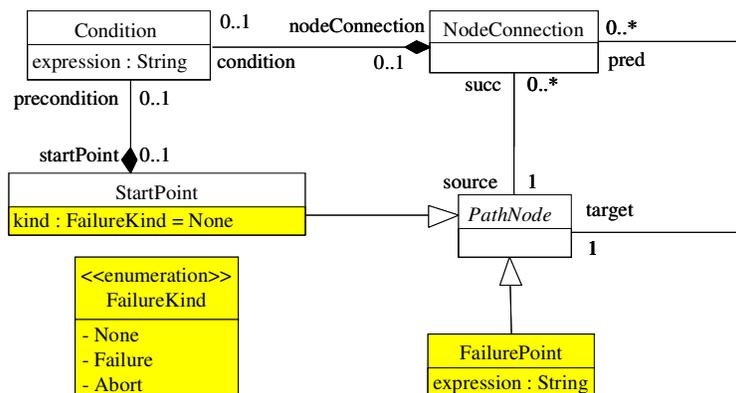


Figure 26 URN Metamodel Extensions: Cancellations and Exceptions

3.7.2 Implicit Failure Points

The *implicit* approach to cancellations and exceptions in UCM 2.0 makes it easier to model cancellations and exceptions that may occur at any point in a scenario. This approach, therefore, does not use failure points to indicate explicitly the location of a failure. The possibility of a failure is simply defined by the existence of a failure or abort path on a map. The failure may occur at any time in the UCM model. The decision when a failure actually occurs is left to scenario definitions.

First, the scenario definition specifies which failure occurs by selecting the failure variable. Then, the scenario definition specifies where the failure may occur by selecting a map, component reference, component definition, responsibility definition, or path node. Note that the occurrence also includes all paths at a lower level that can be reached from the chosen occurrence (i.e., if the failure is chosen to occur on a map, then the failure may also occur on any plug-in map that can be reached via a stub on the map). Also note that specifying a path node is equivalent to the explicit approach but with greater flexibility for the requirements engineer and less clutter on the UCM map itself. Based on the scenario definition, one failure scenario may be created for each possible location where the failure could occur to automatically test the scenario. If this number of locations is too large, heuristics would have to be applied to manage the testing effort. A scaled down test suite could, for example, randomly pick a specified number of the possible locations for the failure tests.

Modifications of the Traversal Mechanism – The traversal mechanism needs to be updated significantly as the structure of scenario definition requires changes for the inclusion of failures. Moreover, new capabilities for the traversal mechanism are needed to generate potentially many failure scenarios for one scenario definition.

Modifications of the URN Metamodel – The changes to the URN metamodel for updated scenario definitions (Figure 27) require the new class `ScenarioFailure` with a composition to `ScenarioDef`. `ScenarioFailure` describes the `failureVariable` (what failure occurred) and the `failureOccurrence` (where the failure occurred).

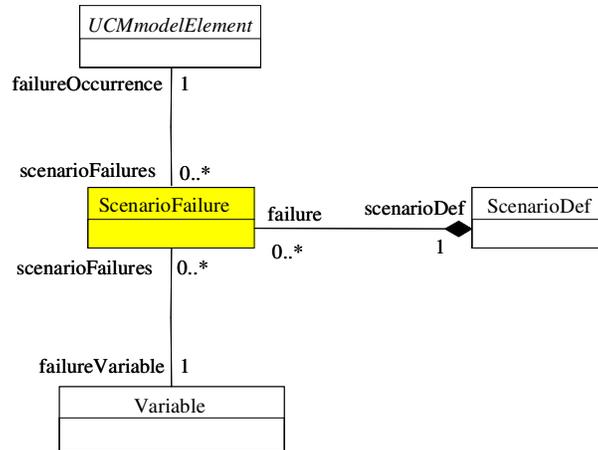


Figure 27 URN Metamodel Extensions: Scenario Failures

3.8. Metamodel of Use Case Maps 2.0

The complete baseline URN metamodel is available in Appendix A: URN Metamodel (Baseline Version). The complete URN metamodel for the ITU draft standard is available in [60] and the URN metamodel as implemented for the jUCMNav tool is available at [143]. Figure 28 and Figure 29 summarize the changes to the URN metamodel. The figures show a simplified URN metamodel with highlighted extensions required for supporting:

- semantic clarifications for UCM 2.0 (synchronizing attribute, blocking attribute, cancelling attribute, replicationFactor attribute, threshold attribute, DeferredChoiceInitialization class),
- singleton maps (singleton attribute),
- component and responsibility plug-in bindings (ComponentBinding class, ResponsibilityBinding class, context attributes), and
- cancellation and exception behavior (FailurePoint class, kind attribute, FailureKind enumeration class, ScenarioFailure class).

In Figure 28, a Stub is a kind of PathNode (which in turn is a URNmodelElement; not shown here) and may be either static (dynamic = false) or dynamic (dynamic = true). Dynamic stubs may be further characterized as synchronizing stubs which in turn may be further characterized as blocking, cancelling, or blocking and cancelling. A static or dynamic stub may have one or many plug-in maps, respectively, that are connected to the

stub by PluginBinding. Each PluginBinding may have a Condition as its precondition (i.e., its selection policy). The replicationFactor of a PluginBinding indicates the number of required instances of the plug-in map.

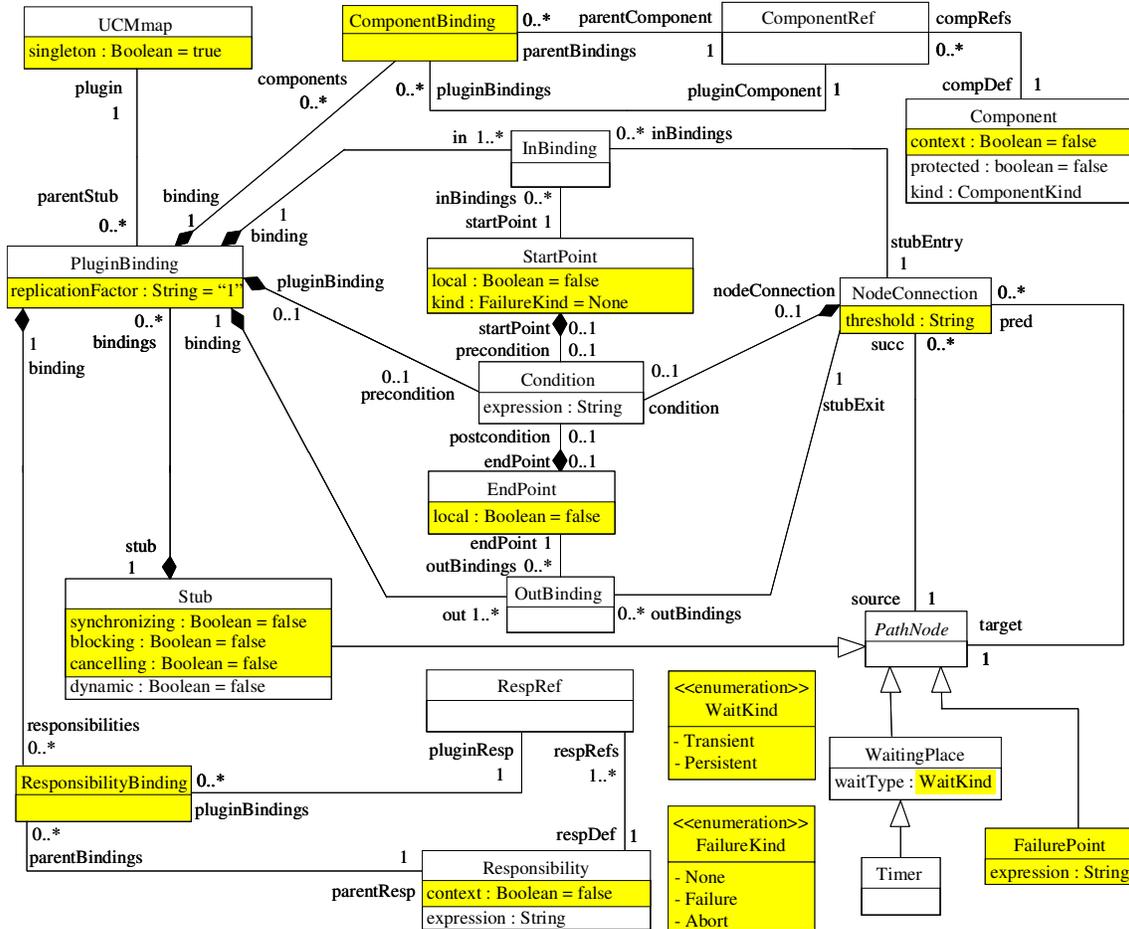


Figure 28 Baseline URN Metamodel with Extensions for UCM 2.0

PluginBindings have InBindings, OutBindings, ComponentBindings, and Responsibility-Bindings. The InBinding of a stub connect the stub’s in-path (the stubEntry NodeConnection) with a StartPoint of the plug-in map. The OutBinding of a stub connect the stub’s out-path (the stubExit NodeConnection) with an EndPoint of the plug-in map. A threshold is defined for a stubExit NodeConnection for out-paths of synchronizing stubs and outgoing branches of an AND-fork. The ComponentBinding of a stub connects components on the parent map (parentComponent) with components on the plug-in map (pluginComponent). The context attribute in Component indicates for which components on a plug-in map a ComponentBinding may exist. Note that components on maps are ref-

erences (ComponentRef) to model-wide definitions of components (Component). The ResponsibilityBinding of a stub connects responsibilities in the model (parentResp) with responsibilities on the plug-in map (pluginResp). The context attribute in Responsibility indicates for which responsibilities on a plug-in map a ResponsibilityBinding may exist. Note that responsibilities on maps are references (RespRef) to model-wide definitions of responsibilities (Responsibility).

A FailurePoint is a new kind of PathNode with one NodeConnection each to the previous and following path element. The expression attribute in the Condition of the NodeConnection to the next path element contains the failure condition of the FailurePoint. The attribute expression of FailurePoint contains the failure expression. A StartPoint has the new attribute kind of type FailureKind. The FailureKind enumeration indicates a regular start point (None), a Failure start point, or an Abort start point. The already existing precondition of a StartPoint captures the guarding condition for failure and abort start points.

StartPoints and EndPoints have the new Boolean attribute local. It differentiates local (true) and regular (false) start and end points.

The waitType attribute of a WaitingPlace (and therefore also of a Timer) is now a WaitKind enumeration, indicating whether the waiting place or timer has Transient or Persistent characteristics.

In Figure 29, a scenario definition (ScenarioDef) consists of the Initializations of Variables, StartPoints, EndPoints, preconditions, and postconditions. Scenario definitions may include other scenario definitions (includedScenarios, parentScenarios), and ScenarioGroup organizes scenario definitions into groups. In addition, ScenarioFailures can now be specified, indicating the failure with its failureVariable and the occurrence of the failure with a UCMmodelElement (i.e., either a PathNode, a UCMmap, a Component, a ComponentRef, or a Responsibility). The DeferredChoiceInitialization is for either OR-forks without conditions or dynamic stubs without selection policies and specifies the deferredChoice (the UCMmodelElement where the deferred choice pattern occurs, i.e., either an OR-fork or a dynamic stub) as well as the enabledBranch (a NodeConnection, i.e., the chosen branch in the OR-fork case) or the enabledPlugin (a PluginBinding, i.e., the chosen plug-in map in the dynamic stub case).

Chapter 4. Assessment of Use Case Maps 2.0

In this chapter, Use Case Maps (UCM) 2.0 is evaluated with the help of a workflow pattern-based approach as described in [127][156] to assess the applicability of UCM 2.0 for workflow/scenario description. UCM 2.0 is analyzed by determining to what extent 43 workflow patterns are directly supported by the notation. A pattern is directly supported if a UCM 2.0 language construct exists that concisely expresses the pattern. More complex, workaround solutions are not taken into account since all patterns can be expressed in some way by standard features of specification languages, including those of UCM 2.0. In other words, the mere ability to express a pattern in some way is not sufficient because conciseness and simplicity are key factors to be considered.

As stated in Section 3.2, the workflow patterns are defined more formally by Colored Petri-Nets. As this assessment provides a mapping from UCM 2.0 language constructs to workflow patterns, UCM 2.0 is defined indirectly also more formally.

A “new” next to a pattern figure indicates a) UCM 2.0 features are not supported by the original UCM notation or b) a specialized, improved traversal mechanism is required to model the workflow pattern. Sections 4.1 to 4.8 go through each of the 43 workflow patterns organized into eight groups. The results of the assessment are then summarized and compared with similar assessments for the Business Process Modeling Notation (BPMN) 1.0, UML 2.0 Activity Diagrams, and the Business Process Execution Language for Web Services (BPEL4WS) 1.1. The requirements for an improved traversal mechanism capable of understanding UCM 2.0 semantics are also listed.

4.1. Basic Control Flow Patterns

WCP-01 Sequence (Figure 30)—This pattern is trivially supported through a UCM path.

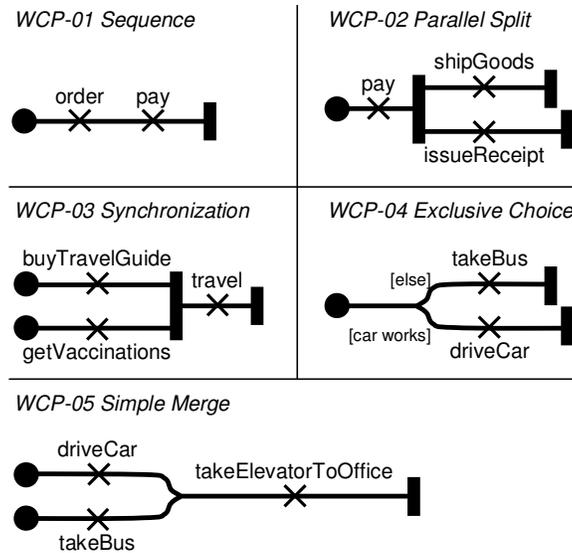


Figure 30 Basic Control Flow Patterns (Group 1)

WCP-02 Parallel Split (Figure 30)—This pattern indicates that several activities can be performed in parallel or in any order. The pattern is mapped directly onto an AND-fork which can have any number of parallel branches.

WCP-03 Synchronization (Figure 30)—This pattern indicates a point in the scenario where several concurrent branches converge into one single branch. The pattern assumes that each incoming branch is executed only once. The pattern is mapped directly onto an AND-join which can have any number of incoming parallel branches. Note that this usage of an AND-join is just a special case of the usage of an AND-join for the WCP-33 Generalized AND-Join pattern described further below.

WCP-04 Exclusive Choice (Figure 30)—This pattern is trivially supported by an OR-fork which can have any number of mutually exclusive branches.

WCP-05 Simple Merge (Figure 30)—This pattern describes a point in the scenario where several alternative branches are merged together into one without synchronization. The pattern is directly supported by an OR-join which can have any number of incoming alternative branches. Note that this usage of an OR-join is just a special case of the usage of an OR-join for the WCP-08 Multi-Merge pattern described further below.

4.2. Advanced Branching and Synchronization Patterns

WCP-06 Multi-Choice (Figure 31)—This pattern describes the situation where more than one alternative branch can be selected and executed at the same time. Since this pattern deals with concurrency without the need for synchronization, it should not be modeled with OR-forks. Extending AND-forks with guards was considered as a possibility for this pattern but abandoned because it complicates the intuitive notion of AND-forks. Therefore, the dynamic stub is the most appropriate way of modeling this pattern. The selection policy enables multiple plug-in maps at the same time (e.g., the selection policy is [Friday] for the first plug-in map and [Friday || Saturday] for the second plug-in map). The pattern is supported by the UCM notation, but only with a change to the current traversal mechanism, i.e., the traversal mechanism of the jUCMNav tool. Currently, the traversal mechanism expects exactly one plug-in map of a dynamic stub to be enabled with an option to choose randomly one out of all enabled plug-in maps in case of a non-deterministic situation.

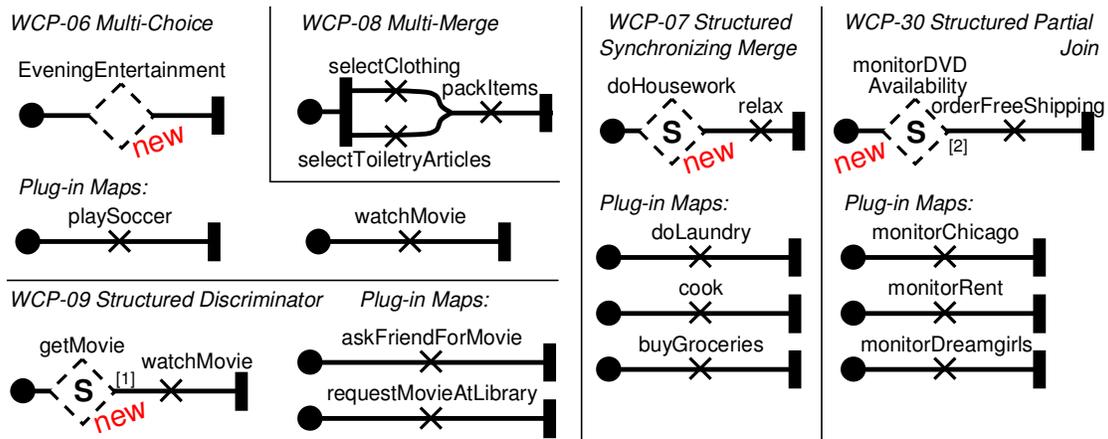


Figure 31 Advanced Branching and Synchronization Patterns – Part I (Group 2)

WCP-07 Structured Synchronizing Merge (Figure 31)—This pattern describes a point in the scenario where several branches are merged together into one with synchronization. The pattern assumes that each incoming branch is at the most executed once. It may be assumed that the number of branches taken is known at the time the merge is reached (at least one branch and at the most all branches). The pattern is modeled with a synchronizing stub and one plug-in map for each branch. The selection policy of the synchronizing stub enables the required number of plug-in maps (e.g., the selection

policy is [Wednesday || Saturday] for the first plug-in map, [any day] for the second plug-in map, and [Saturday || Sunday] for the third plug-in map). The traversal mechanism then waits until all started plug-in maps have completed their activities before continuing along the path after the stub. jUCMNav’s current traversal mechanism, however, needs to be updated to support synchronizing stubs (therefore any pattern that uses a synchronizing stub will have the “new” indicator). Note that this pattern was previously called Synchronizing Merge.

WCP-08 Multi-Merge (Figure 31)—This pattern covers the situation where multiple branches are merged into one without synchronization but with the expectation that activities following the merge will be performed once for each active branch. The pattern is directly supported by an OR-join. Note that OR-constructs and AND-constructs in UCM do not have to be properly nested and may therefore be used together.

WCP-30 Structured Partial Join (Figure 31)—This pattern is a type of merge with multiple concurrent incoming branches. The pattern is useful in a case where the n^{th} branch out of m incoming branches triggers the continuation of the scenario. All other incoming branches are ignored. Upon receipt of the last incoming branch, the Structured Partial Join is reset, so that the next set of incoming branches can once again trigger the continuation of the scenario. This pattern is modeled with a synchronizing stub for which the synchronization threshold is specified (e.g., three DVDs are monitored for availability and two DVDs are sufficient to receive free shipping – the synchronization threshold 2 is indicated on the stub’s out-path). The selection policy of the stub enables all plug-in maps. The traversal mechanism continues with the scenario once the n^{th} plug-in map has completed. All other plug-in maps do not trigger a continuation of the scenario. Once the third plug-in map has also completed, the stub is reset. Note that this pattern was previously called N-out-of-M Join.

WCP-31 Blocking Partial Join—This pattern is a special case of the Structured Partial Join in that no plug-in map can be activated again until the stub is reset. It is directly supported by the blocking attribute of the synchronizing stub. The example of the Structured Synchronizing Merge in Figure 31 is a good candidate for this pattern, if only two out of the three plug-in maps need to finish and only one household can be dealt with at a time due to resource constraints.

WCP-32 Cancelling Partial Join—This pattern is a special case of the Structured Partial Join in that all remaining plug-in maps are cancelled once the synchronization threshold is reached and before resetting the stub. It is directly supported by the cancelling attribute of the synchronizing stub. The example of the Structured Partial Join in Figure 31 is a good candidate for this pattern as monitoring of the third DVD can be cancelled once two other DVDs have become available.

WCP-09 Structured Discriminator (Figure 31)—This pattern is a special case of the Structured Partial Join. It is a 1-out-of-m join and can be modeled similarly to the Structured Partial Join by a synchronizing stub with the synchronization threshold set to 1 (as indicated on the stub's out-path). Note that this pattern was previously called Discriminator.

WCP-28 Blocking Discriminator—This pattern is a special case of the Structured Discriminator in that no plug-in map can be activated again until the stub is reset. It is directly supported by the blocking attribute of the synchronizing stub. The example of the Structured Synchronizing Merge in Figure 31 is a good candidate for this pattern, if only one out of the three plug-in maps need to finish and only one household can be dealt with at a time due to resource constraints.

WCP-29 Cancelling Discriminator—This pattern is a special case of the Structured Discriminator in that all remaining plug-in maps are cancelled once the first plug-in map finishes and before resetting the stub. It is directly supported by the cancelling attribute of the synchronizing stub. The example of the Structured Discriminator in Figure 31 is a good candidate for this pattern as either the library request or asking a friend can be cancelled once the other is successful.

WCP-33 Generalized AND-Join (Figure 32)—This pattern indicates a point in the scenario where several concurrent branches converge into one single branch. The pattern does not assume a structured context. The pattern is mapped directly onto an AND-join which can have any number of incoming parallel branches each of which can be activated multiple times (e.g., a factory assembling a rudimentary computer once a motherboard, hard disk, and tower are available). The traversal mechanism remembers for each incoming branch how often the incoming branch arrived at the AND-join.

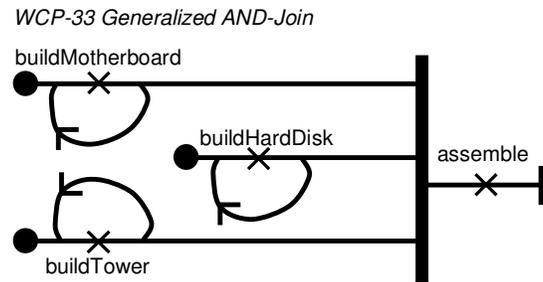


Figure 32 Advanced Branching and Synchronization Patterns – Part II (Group 2)

WCP-37 Local Synchronizing Merge—This pattern describes a synchronization behavior in an unstructured context that determines whether to proceed with the scenario based on information locally available to the merge construct. It could be supported if path traversal were allowed to continue from an end point on a plug-in map that is not connected to any of the synchronizing stub’s out-paths. As this is not desirable in the context of UCM, the pattern is not supported by the UCM 2.0 notation. Note that this pattern was previously called Acyclic Synchronizing Merge.

WCP-38 General Synchronizing Merge—This pattern describes a synchronization behavior in an unstructured context that determines whether to proceed with the scenario based on information not available locally to the merge construct. The pattern is not supported by the UCM 2.0 notation as it requires the traversal mechanism to perform an evaluation of possible future states to decide whether the path traversal is allowed to continue.

WCP-41 Thread Merge and **WCP-42 Thread Split**—These two patterns describe multiple execution threads that are spawned and merged on a single branch. They are not supported by the UCM 2.0 notation. The patterns could be supported with specialized AND-fork and AND-join constructs, allowing only one outgoing or incoming branch. Furthermore, a new attribute of AND-forks must then be able to specify how many threads should be created, i.e., a replication factor, and a new attribute for AND-joins must then be able to specify how many threads have to arrive before the traversal can continue, i.e., a synchronization threshold.

4.3. Multiple Instance Patterns

WCP-12 Multiple Instances without Synchronization (Figure 33)—This pattern describes the situation where an activity in a scenario is spawned off, i.e., it needs to be executed multiple times in parallel without the need to synchronize any instances of the activity. The pattern is directly supported by UCM 2.0 with a dynamic stub with only one plug-in map for which a replication factor is defined. This pattern can also be modeled easily with an AND-fork and a loop. Even though this is less concise than using the dynamic stub, the pattern is deemed to be directly supported by the UCM notation, because the same approach is used in the assessment for UML AD [129], and this AND-fork / loop combination is observed in many examples from the UCM literature.

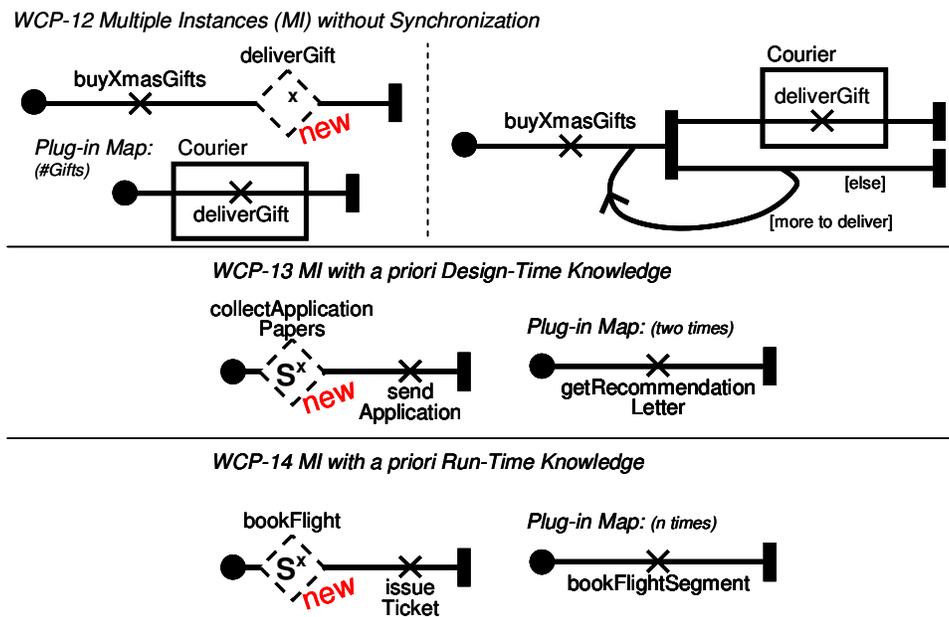


Figure 33 Multiple Instance Patterns (Group 3)

WCP-13 Multiple Instances with a priori Design-Time Knowledge (Figure 33)—This pattern describes a point in the scenario where several instances of an activity have to be executed in parallel. These instances are synchronized in that the scenario continues only when all instances are completed. The number of instances is known at design time. The pattern is directly supported by UCM 2.0 with a synchronizing stub with only one plug-in map for which a replication factor is defined.

WCP-14 Multiple Instances with a priori Run-Time Knowledge (Figure 33)—

This pattern is the same as the previous one except that the number of instances is not known at design time but at runtime before the instances have to be created. Again, the pattern is directly supported by a synchronizing stub with one plug-in map replicated the desired number of times as known at this point in the scenario. This replication is achieved by dynamically setting a scenario variable to the desired number of instances before the stub is reached (e.g., in an expression of a responsibility before the stub or in the initialization of a scenario definition).

WCP-15 Multiple Instances without a priori Run-Time Knowledge—This pattern is also very similar to the previous two patterns. The difference is that the number of instances is not known, not even while the instances are executing (i.e., a new instance may be created when other instances are already executing). An example of this pattern is a court case with a `callWitness` activity. New witnesses may be called during the court case even when other witnesses already have been questioned. Only when all witnesses have been heard will the jury start deliberation. The pattern could be modeled similarly to the previous pattern but requires additional processing by the traversal mechanism in order to decide whether another plug-in map instance is still required. This processing could be in the form of a condition that needs to be evaluated. The evaluation would have to take place while already existing plug-in maps are executing. Moreover, new instances of a plug-in map would have to be created while the stub is already executing instead of when reaching the stub the first time. As this moves considerably further away from a key assumption about the behavior of stubs, this pattern is not supported by UCM 2.0.

WCP-34 Static Partial Join for Multiple Instances—This pattern is the same as WCP-13/14 but without the constraint that all plug-in maps have to complete. The pattern is directly supported by the synchronization threshold of synchronizing stubs. Only as many plug-in maps as defined by the synchronization threshold have to complete. The example of WCP-13 in Figure 33 is a good candidate for this pattern if more requests for recommendation letters are sent out than needed for the application (i.e., the replication factor is greater than the synchronization threshold).

WCP-35 Cancelling Partial Join for Multiple Instances—This pattern adds the ability to abort remaining plug-in maps once the required number of plug-in maps have

completed. It is directly supported by the cancelling attribute of the synchronizing stub. The example described for WCP-34 is a good candidate for this pattern.

WCP-36 Dynamic Partial Join for Multiple Instances—This pattern is the same as WCP-15 without the constraint that all plug-in maps need to complete. As WCP-15 is not supported, this pattern is therefore also not supported by UCM 2.0.

4.4. State-Based Patterns

WCP-16 Deferred Choice (Figure 34)—This pattern describes a situation very similar to WCP-04 Exclusive Choice. The crucial difference is that the decision regarding which branch to choose is not made at the choice point but implicitly by starting the first responsibility (or other path element) of one branch. In other words, all alternatives are possible until one alternative starts at which point all other alternatives are not available anymore (the other alternatives do not even start). An example for this pattern is the approval of a document which could be done either by the head of the department or by the project manager. The pattern is modeled with UCM with the help of an OR-fork (or a dynamic stub with plug-in maps) without any specified conditions. The pattern is therefore directly supported but requires the traversal mechanism to understand such OR-forks or dynamic stubs.

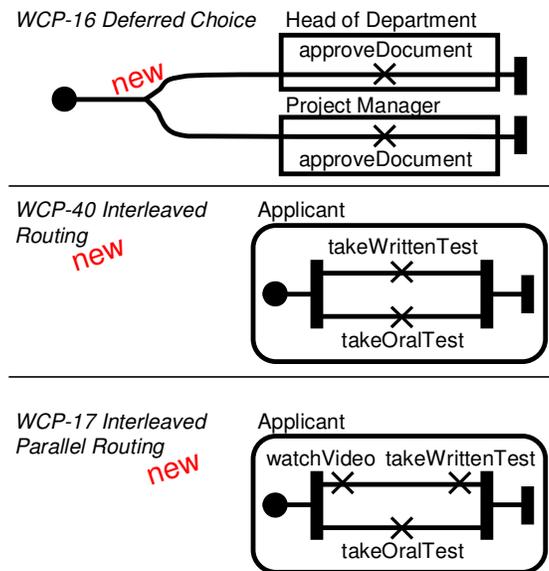


Figure 34 State-Based Patterns – Part I (Group 4)

While the specification of this pattern is rather straightforward, the implementation may be complex depending on what techniques and formalisms are used following UCM in the software development process. The reason is that distributed decision making may be required since all alternatives have to be offered or monitored to find out which one is chosen. UCM deals with this complexity by requiring the scenario definitions to simply specify explicitly which branch or plug-in map is chosen. This approach still allows scenarios with deferred choices to be traversed and tested by the traversal mechanism. The UCM scenario definition, however, must be updated to support this pattern.

WCP-40 Interleaved Routing (Figure 34)—This pattern describes the situation where two or more activities may be performed in any order but not in parallel. All examples given in [156][158] indicate a resource conflict as the reason for not being able to execute the activities in parallel. Therefore, UCM with a specialized traversal mechanism could directly support the pattern. The traversal mechanism can determine from the binding of path nodes to components whether concurrent responsibilities (or any other path nodes) are bound to the same component. If this is the case and if the component is of kind object, then the concurrent branches are interpreted as Interleaved Routing. All other concurrent paths are interpreted as strictly parallel.

WCP-17 Interleaved Parallel Routing (Figure 34)—This pattern is similar to WCP-40 Interleaved Routing and is therefore supported through the same mechanism as WCP-40 Interleaved Routing. The difference is that this pattern introduces partial ordering into the interleaved activities.

WCP-39 Critical Section—This pattern is supported in the UCM metamodel by the attribute protected in Component. jUCMNav's current traversal mechanism, however, does not use this attribute and therefore has to be extended in order to ensure that another path cannot enter a protected component if one path is already being traversed in the component.

WCP-18 Milestone (Figure 35)—This pattern describes a point in the scenario where activity B can be executed (possibly multiple times) because activity A has already been executed and activity C has not yet been executed. The pattern is similar to Deferred Choice in that there is a race condition between two activities (B and C in this case) and only one activity is executed at one time. Milestone is different than Deferred Choice in

that activity C is executed at some point. UCM can model the Milestone pattern with an OR-fork without conditions and an implicit loop. The example shows `openBidding` as activity A, `bid` as activity B, and `closeBidding` as activity C. A specialized traversal mechanism, however, is required for this pattern just as it is required for Deferred Choice. A more complicated variation of the Milestone pattern occurs when one parallel branch influences another parallel branch. In this case, variables in addition to the deferred choice are required to indicate to the second branch (e.g., `sendBill` and `receivePayment`) when the milestone is reached in the first branch (e.g., `shipGoods`). Essentially, only after goods are shipped (i.e., the variable `Shipped` is set to true) and between sending a bill and receiving payment is it possible to resend the bill. The complexity of the model for the parallel variation of the pattern results in a partial score of support for this pattern.

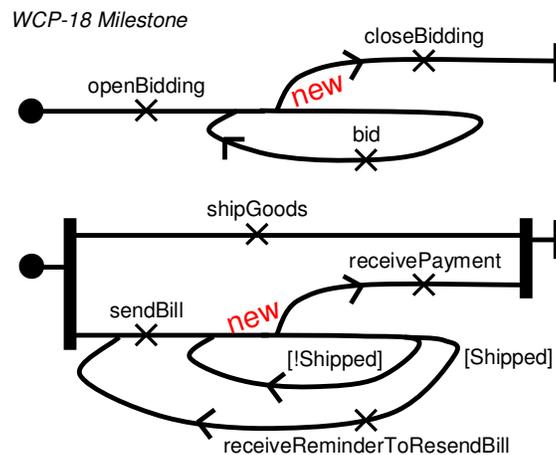


Figure 35 State-Based Patterns – Part II (Group 4)

4.5. Cancellation and Force Completion Patterns

WCP-19 Cancel Task (Figure 36)—This pattern describes the situation where an individual activity is cancelled. This pattern is supported by the general cancellation and exception mechanism of the UCM 2.0 notation. The traversal mechanism for UCM models, however, does not have the notion of duration for individual responsibilities. Therefore, a responsibility is an atomic action that cannot be interrupted. The responsibility, however, can be cancelled before it is started. An interruption of a responsibility is modeled simply by a failure point after the responsibility. In this case, context information

such as the naming of the path elements indicates that the responsibility is a long-running activity and is interrupted (alternatively, two responsibilities – the first indicating the beginning of the activity and the second indicating the end of the activity – could be shown with the failure point in the middle). The example shows that monitorPatient is interrupted by the monitoringAlarm failure point (alternatively, the responsibility endMonitorPatient could be shown after the failure point). The failure results in emergency treatment before the monitoring continues. In any case, jUCMNav’s traversal mechanism needs to be updated to support failure points, failure paths, and abort paths, therefore any pattern that uses these concepts will have the “new” indicator. Note that this pattern was previously called Cancel Activity.

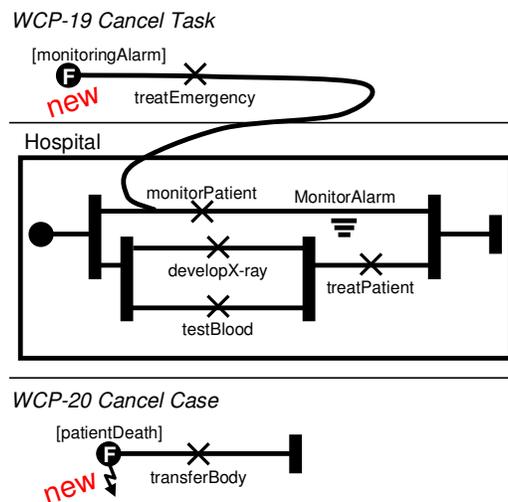


Figure 36 Cancellation and Force Completion Patterns – Part I (Group 5)

WCP-20 Cancel Case (Figure 36)—This pattern describes the situation where a complete scenario is cancelled. This pattern is supported by the general cancellation and exception mechanism of the UCM 2.0 notation as an abort may be defined at the highest level in the map hierarchy for the scenario. The example shows that the traversal of the whole hospital scenario stops when the abort patientDeath occurs.

WCP-25 Cancel Region—This pattern describes the situation where a portion of the scenario is cancelled. This pattern is again supported by the general cancellation and exception mechanism of the UCM 2.0 notation, because an abort may be scoped to a plug-in map that describes the portion of the scenario that should be cancelled.

WCP-26 Cancel Multiple Instance Activity (Figure 37)—This pattern describes the situation where an activity with multiple instances is cancelled. This pattern is supported by the general cancellation and exception mechanism of the UCM 2.0 notation, as abort paths can be defined on a map that models multiple instance activities (e.g., with a synchronizing stub for which a replicated plug-in map is defined). The scoping feature of the abort path ensures that all instances are cancelled. The example shows that the remaining lab experiments are cancelled once the lab booking expires and that the report is not written.

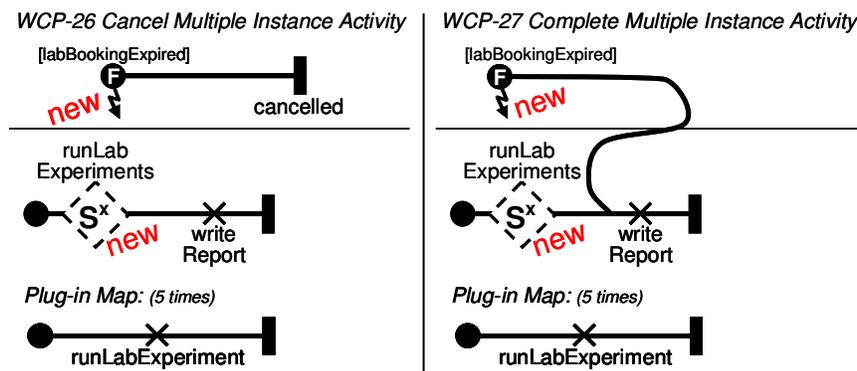


Figure 37 Cancellation and Force Completion Patterns – Part II (Group 5)

WCP-27 Complete Multiple Instance Activity (Figure 37)—This pattern is similar to WCP-26 Cancel Multiple Instance Activity but allows the scenario to continue after the cancellation of the remaining multiple instances. The example shows that the remaining lab experiments are cancelled once the lab booking expires, but that the scenario continues and the report is written. This pattern is also similar to WCP-35 Cancelling Partial Join for Multiple Instances. The difference is that WCP-35 may cancel remaining multiple instances only if the synchronization threshold is reached whereas this pattern may cancel remaining multiple instances for other reasons not related to the synchronization threshold.

4.6. Iteration Patterns

WCP-10 Arbitrary Cycles (Figure 38)—This pattern addresses non-structured cycles. The pattern is directly supported by the UCM notation since loops created with OR-forks and OR-joins do not have to be properly nested.

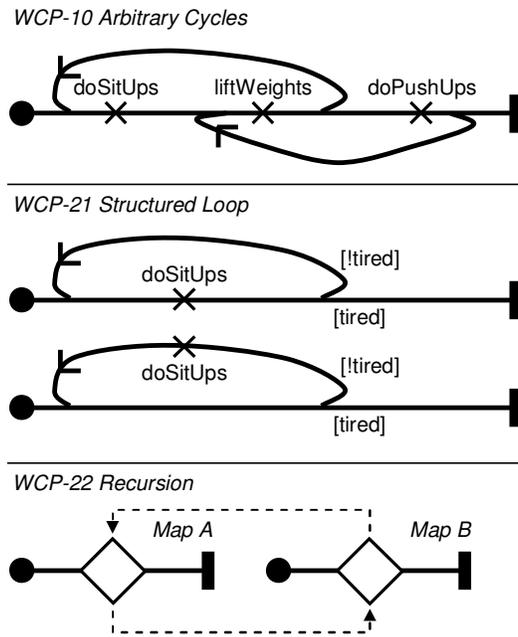


Figure 38 Iteration Patterns (Group 6)

WCP-21 Structured Loop (Figure 38)—This pattern is directly supported by the UCM notation since properly nested loops can be created with OR-forks and OR-joins. The example shows the looping condition being examined either at the end or the beginning of the loop.

WCP-22 Recursion (Figure 38)—This pattern describes the ability to recursively invoke an activity. It is supported by the UCM notation as plug-in maps may be assigned to a stub in a recursive fashion.

4.7. Termination Patterns

WCP-11 Implicit Termination—This pattern indicates that a scenario terminates automatically if there is nothing left to do. The pattern is directly supported by the UCM notation since there is no need to explicitly specify a termination responsibility in a UCM model. All active branches having reached an end point that is not connected to a parent map indicates the end of a scenario.

WCP-43 Explicit Termination—This pattern allows for explicit termination of a scenario, typically indicated by a specific end node. It is not supported by the UCM 2.0 notation although a simple workaround solution exists with failure points and abort paths.

4.8. Trigger Patterns

WCP-23 Transient Trigger and **WCP-24 Persistent Trigger** (Figure 39)— These two patterns are modeled using the waiting place and timer constructs (e.g., by connecting an end point or path to a waiting place or timer). The `waitType` attribute of a waiting place or timer differentiates between transient and persistent triggers, but there is no visual indication for the `waitType`. The example shows that a library clerk will answer the phone only during the work shift, but will check in all items even if they are dropped off after hours. Unfortunately, jUCMNav’s current traversal mechanism does not take the `waitType` into account and persists only the first triggers.

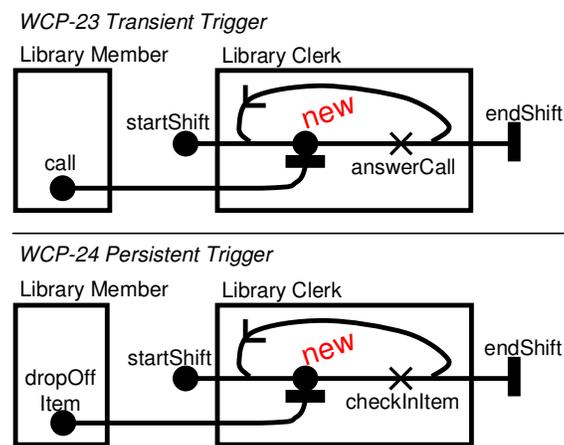


Figure 39 Trigger Patterns (Group 8)

4.9. Assessment Results

A number of patterns assume a structured context ensuring that all incoming branches of a modeling construct are not triggered multiple times before all incoming branches have arrived once at the construct. These patterns are WCP-03, WCP-05, WCP-07, WCP-09, WCP-30, WCP-34, and WCP-37. This assumption is not so much an assumption about the modeling notation but about the model itself. UCM can be used to describe structured and non-structured models. Once a particular modeling type is chosen, the semantics of the UCM 2.0 constructs that support the patterns as summarized in Table 2 fulfills the criteria set out in [127].

In the last four columns of Table 4 and the Score columns of Table 2 and Table 3, + denotes support of the notation for the pattern, +/- denotes partial support, and - denotes

no support. The * in the fourth column of Table 2 indicates that the pattern is supported only by an improved UCM traversal mechanism and not the currently available traversal mechanism of jUCMNav. An X in the fourth column of Table 2 indicates that the pattern is still not supported by the latest version of the URN standard, i.e., all patterns with a * but not an X in the fourth column are supported by the latest version of the URN standard with its improved traversal mechanism [60]. The following abbreviations are used in Table 2: BA (Blocking Attribute), CA (Cancelling Attribute), DTK (Design-Time Knowledge), MI (Multiple Instance(s)), RPM (Replicated Plug-in Map), RTK (Run-Time Knowledge), and ST (Synchronization Threshold).

Table 3 summarizes the results of the assessment whereas Table 4 gives a detailed comparison of UCM 2.0 with the Business Process Modeling Notation (BPMN) 1.0 [109], UML 2.0 Activity Diagrams (AD) [112], and the Business Process Execution Language for Web Services (BPEL4WS) 1.1 [34]. The assessments of these three standards are taken from [127]. Table 3 also shows the assessment result for the latest version of the URN standard [60]. The performance of the UCM 2.0 notation is excellent compared to the three other modeling notations when all 43 patterns are taken into account. Only seven patterns (WCP-15, WCP-36, WCP-37, WCP-38, WCP-41, WCP-42, and WCP-43) remain unsupported and one partially supported (WCP-18). Three out of the seven unsupported patterns are not supported by any of the compared notations (WCP-15, WCP-36, WCP-38), while the partially supported pattern is not supported by the other notations. Only eleven patterns are supported by all notations. These are the patterns from the basic control-flow group (WCP-01 to WCP-05), WCP-06 Multi-Choice, WCP-11 Implicit Termination, WCP-12 Multiple Instances without Synchronization, WCP-19 Cancel Task, WCP-20 Cancel Case, and WCP-21 Structured Loop.

Table 2 Support of UCM 2.0 Notation for Workflow Patterns

Workflow Pattern	Score	UCM 2.0 Construct	Notes
WCP-01 Sequence	+	Path	
WCP-02 Parallel Split	+	AND-fork	
WCP-03 Synchronization	+	AND-join	
WCP-04 Exclusive Choice	+	OR-fork	
WCP-05 Simple Merge	+	OR-join	
WCP-06 Multi-Choice	+	Dynamic stub	*
WCP-07 Structured Synchr. Merge	+	Synchronizing stub	*
WCP-08 Multi-Merge	+	OR-join	
WCP-09 Structured Discriminator	+	Synchronizing stub with ST	*

Workflow Pattern	Score	UCM 2.0 Construct	Notes
WCP-10 Arbitrary Cycles	+	Non-nested OR-forks and OR-joins	
WCP-11 Implicit Termination	+	End point	
WCP-12 MI without Synchronization	+	AND-fork and OR-fork	
WCP-13 MI with a priori DTK	+	Synchronizing stub with RPM	*
WCP-14 MI with a priori RTK	+	Scenario variables and synch. stub with RPM	*
WCP-15 MI without a priori RTK	-	---	
WCP-16 Deferred Choice	+	OR-fork or dynamic stub without conditions	* X
WCP-17 Interleaved Parallel Routing	+	Concurrent branches in one component	*
WCP-18 Milestone	+/-	Scenario variables and deferred choice	*
WCP-19 Cancel Task	+	Failure/abort path	* X
WCP-20 Cancel Case	+	Failure/abort path	* X
WCP-21 Structured Loop	+	Properly nested OR-fork and OR-join	
WCP-22 Recursion	+	Recursive plug-in map	
WCP-23 Transient Trigger	+	waitType attribute of waiting place/timer	*
WCP-24 Persistent Trigger	+	waitType attribute of waiting place/timer	*
WCP-25 Cancel Region	+	Failure/abort path	* X
WCP-26 Cancel MI Activity	+	Failure/abort path and syn. stub with RPM	* X
WCP-27 Complete MI Activity	+	Failure/abort path and syn. stub with RPM	* X
WCP-28 Blocking Discriminator	+	Synchronizing stub with ST and BA	*
WCP-29 Cancelling Discriminator	+	Synchronizing stub with ST and CA	* X
WCP-30 Structured Partial Join	+	Synchronizing stub with ST	*
WCP-31 Blocking Partial Join	+	Synchronizing stub with ST and BA	*
WCP-32 Cancelling Partial Join	+	Synchronizing stub with ST and CA	* X
WCP-33 Generalized AND-Join	+	AND-join	
WCP-34 Static Partial Join for MI	+	Synchronizing stub with RPM and ST	*
WCP-35 Cancelling Part. Join for MI	+	Synchronizing stub with RPM, ST, and CA	* X
WCP-36 Dynamic Partial Join for MI	-	---	
WCP-37 Local Synchronizing Merge	-	---	
WCP-38 General Synchr. Merge	-	---	
WCP-39 Critical Section	+	Protected component	*
WCP-40 Interleaved Routing	+	Concurrent branches in one component	*
WCP-41 Thread Merge	-	---	
WCP-42 Thread Split	-	---	
WCP-43 Explicit Termination	-	---	

Table 3 Summary of Assessment of all 43 Workflow Patterns

Score	UCM 2.0	Latest Version	BPMN	AD	BPEL4WS
+	35	26	24	25	17
+/-	1	1	9	5	4
-	7	16	10	13	22

Table 4 Comparison of UCM 2.0, BPMN, UML 2.0 AD, and BPEL4WS

Workflow Pattern	UCM	BPMN	UML AD	BPEL4WS
WCP-01 Sequence	+	+	+	+
WCP-02 Parallel Split	+	+	+	+
WCP-03 Synchronization	+	+	+	+
WCP-04 Exclusive Choice	+	+	+	+
WCP-05 Simple Merge	+	+	+	+
WCP-06 Multi-Choice	+	+	+	+
WCP-07 Structured Synchronizing Merge	+	+	-	+

Workflow Pattern	UCM	BPMN	UML AD	BPEL4WS
WCP-08 Multi-Merge	+	+	+	-
WCP-09 Structured Discriminator	+	+/-	+/-	-
WCP-10 Arbitrary Cycles	+	+	+	-
WCP-11 Implicit Termination	+	+	+	+
WCP-12 Multiple Instances (MI) without Synchronization	+	+	+	+
WCP-13 MI with a priori Design-Time Knowledge	+	+	+	-
WCP-14 MI with a priori Run-Time Knowledge	+	+	+	-
WCP-15 MI without a priori Run-Time Knowledge	-	-	-	-
WCP-16 Deferred Choice	+	+	+	+
WCP-17 Interleaved Parallel Routing	+	-	-	+/-
WCP-18 Milestone	+/-	-	-	-
WCP-19 Cancel Task	+	+	+	+
WCP-20 Cancel Case	+	+	+	+
WCP-21 Structured Loop	+	+	+	+
WCP-22 Recursion	+	-	-	-
WCP-23 Transient Trigger	+	-	+	-
WCP-24 Persistent Trigger	+	+	+	+
WCP-25 Cancel Region	+	+/-	+	+/-
WCP-26 Cancel MI Activity	+	+	+	-
WCP-27 Complete MI Activity	+	-	-	-
WCP-28 Blocking Discriminator	+	+/-	+/-	-
WCP-29 Cancelling Discriminator	+	+	+	-
WCP-30 Structured Partial Join	+	+/-	+/-	-
WCP-31 Blocking Partial Join	+	+/-	+/-	-
WCP-32 Cancelling Partial Join	+	+/-	+	-
WCP-33 Generalized AND-Join	+	+	-	-
WCP-34 Static Partial Join for MI	+	+/-	-	-
WCP-35 Cancelling Partial Join for MI	+	+/-	-	-
WCP-36 Dynamic Partial Join for MI	-	-	-	-
WCP-37 Local Synchronizing Merge	-	-	+/-	+
WCP-38 General Synchronizing Merge	-	-	-	-
WCP-39 Critical Section	+	-	-	+
WCP-40 Interleaved Routing	+	+/-	-	+
WCP-41 Thread Merge	-	+	+	+/-
WCP-42 Thread Split	-	+	+	+/-
WCP-43 Explicit Termination	-	+	+	-

Semantics is the biggest concern with regards to modeling scenarios with UCM. UCM 2.0 clarifies the semantics by describing more precisely the expected behavior of path and structural elements when the UCM 2.0 model is traversed given a scenario definition. In summary, the UCM 2.0 traversal mechanism has to deal with the following situations:

- Support OR-forks without any conditions.
- Support dynamic stubs without any selection policy.
- Support dynamic stubs where the conditions of several plug-in maps evaluate to true and parallel execution of those plug-in maps.

- Support synchronizing stubs with synchronization thresholds, replicated plug-in maps, blocking attributes, and cancelling attributes.
- Support protected components.
- Identify interleaving based on the binding of path nodes to components of kind object.
- Support failure points, failure paths, and abort paths.
- Support local start and end points.
- Support wait types of waiting places and timers.
- Support singleton maps and non-singleton maps.
- Support component and responsibility plug-in bindings.

Appendix B: UCM 2.0 Traversal Mechanism provides detailed requirements for the improved traversal mechanism for UCM 2.0.

4.10. Summary

This chapter evaluates the Use Case Map (UCM) 2.0 notation with the help of a structured and more formal assessment based on a set of workflow patterns. The assessment concluded that UCM 2.0 is a competitive scenario language compared to BPMN 1.0, BPEL4WS 1.1, and UML 2.0 Activity Diagrams. This result is particularly interesting because UCM offers additional benefits compared to these three standards. Foremost, UCM is integrated with GRL goal models in URN. Secondly, UCM offers greater flexibility for connecting sub-diagrams and for modeling component containment. UCM also integrates a simple data model, performance annotations, and a simple action language used for analysis. The assessment highlights requirements for an improved traversal mechanism for UCM 2.0 that further standardizes the semantics of UCM models and enables full support of most generic workflow patterns. Given the evaluated extensions to the UCM notation, the next chapter now gives an intuitive, example-based introduction to the Aspect-oriented User Requirements Notation (AoURN), while Chapter 6 then discusses the core of AoURN.

Chapter 5. AoURN in a Nutshell

The purpose of this chapter is to give an intuitive introduction to the Aspect-oriented User Requirements Notation (AoURN) with a simple example that covers the basic concepts of the notation. Therefore, AoURN model elements are introduced in this chapter without giving definitions for them while the following chapter will then give definitions of all introduced model elements. First, two maps of the example system, an Online Video Store, are described and the Logging concern is composed with them. Later in the chapter, a goal model for the example system is described and the Logging concern is again composed with it. The same is repeated for two other concerns before interactions between the concerns are discussed.

5.1. Online Video Store

The example describes an Online Video Store (OVS) that allows Customers to order movies online and the General Public to browse through the store's movie encyclopedia. Figure 40 shows two main scenarios of the system, Order Movie to the left and Browse Movie to the right. For the Order Movie scenario, the Customer first selects a movie, then the OVS processes the order, the Customer pays for the movie, and finally the OVS sends the movie to the customer. For the Browse Movie scenario, the General Public first selects a movie and the OVS retrieves a summary of the movie. The General Public may then ask for further details in which case the OVS provides them.

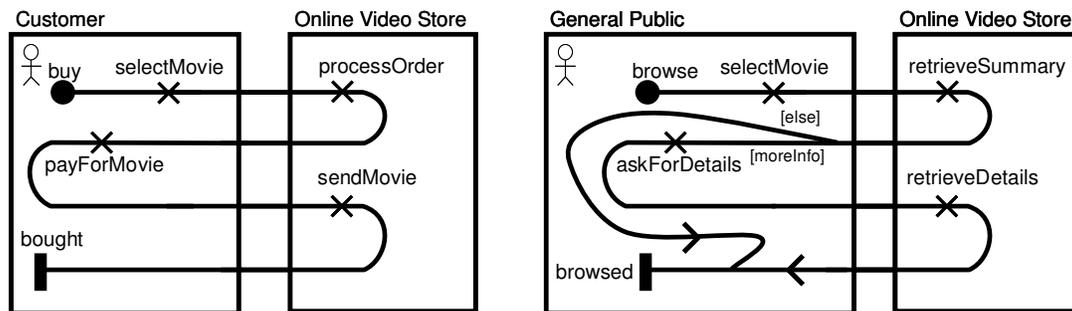


Figure 40 Two Main Scenarios of the Online Video Store System

5.2. Logging Concern (Scenario)

Three crosscutting concerns are to be added to the OVS with the help of aspect-oriented techniques to ensure that the concerns are properly encapsulated. The three concerns are Logging, Authentication, and Communication. When adding crosscutting concerns to a system, one needs to first think about the concern's aspectual properties by asking the question "what is the concern?" For the Logging concern, the answer to this question entails the logging of information with the help of a **Logger** as illustrated in Figure 41.a.

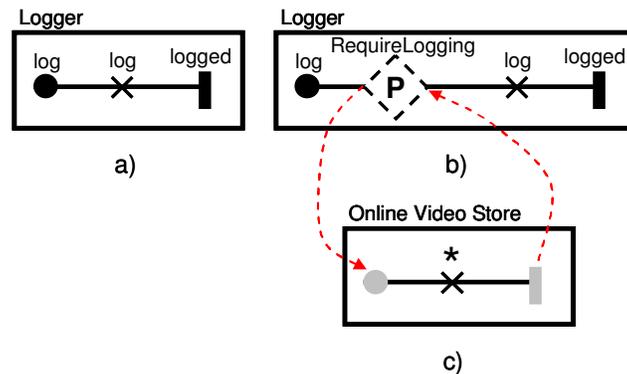


Figure 41 Step by Step Modeling of the Logging Concern (Scenarios)

The next step is to think about the composition rules for the Logging concern. In this case, logging is to be done after the system performed an operation. Therefore, the *pointcut stub* (\textcircled{P} , Figure 41.b) for the logger is added to the model, so that the log responsibility occurs after the pointcut stub (Figure 41.b). The causal relationship between the pointcut stub and the aspectual properties clearly visualizes the "after" composition rule. The pointcut stub by itself does not define where the crosscutting concern is to be applied. It defines how the crosscutting concern is to be applied, simply acting as a placeholder for those locations in the model where the crosscutting concern is to be applied.

The third step actually identifies the locations in the main scenarios by specifying a pattern for those locations that require logging. The pattern is described on its own map called the *pointcut map* (Figure 41.c). In the case of the Logging concern, any responsibility of the Online Video Store needs to be logged and hence captured by the pattern. Therefore, the pattern matches any responsibility, indicated by the wildcard *, in the Online Video Store component. The specified pattern is not applicable only to the Logging concern, but may be reused by any other concern that affects the same locations. To

indicate that the pattern is to be used by the Logging concern, the pointcut map needs to be plugged into the pointcut stub of the Logging concern, as indicated by the plug-in bindings connecting the pointcut stub's in-path and out-path with the pointcut map's start point and end point, respectively. The scenario model for the Logging concern is now fully specified.

For any aspect-oriented technique to be practical, it must be possible to examine the impact of applying a crosscutting concern to a model. In the case of the Logging concern, the pattern specified by the pointcut map in Figure 41.c matches against the four responsibilities of the Online Video Store in Figure 40. AoURN indicates the impact of a concern by automatically placing *aspect markers* (◆, Figure 42) at the appropriate locations in the model as defined by the pointcut map and the composition rule. Hence, an aspect marker for the Logging concern is added after each of the four responsibilities in the Online Video Store component, indicating that the pattern of the Logging concern is matched four times.

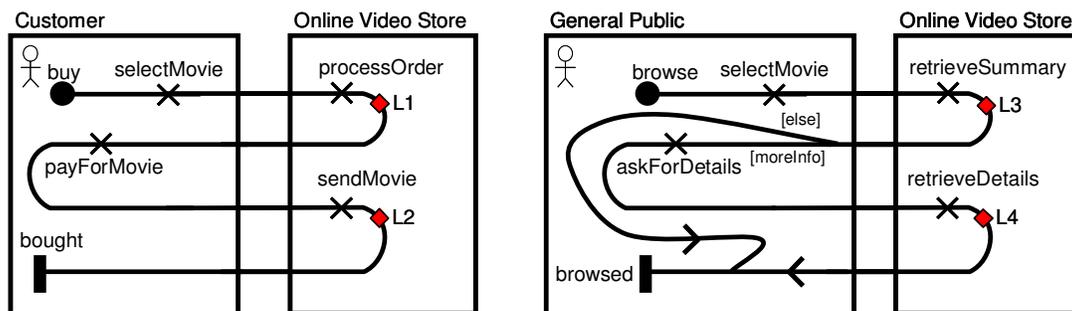


Figure 42 Impact of Logging Concern on OVS System (Scenario)

While aspect markers are useful for the requirements engineer since they indicate active concerns, more details are required to fully understand the impact of a concern on the model. It must be possible to examine the actual aspectual properties that are inserted by the concern at the aspect marker. Therefore, AoURN supports *AoViews* as illustrated for one aspect marker in Figure 43. An *AoView* is a visualization of the aspectual properties inserted at the location of the aspect marker. In addition to automatically inserting aspect markers, AoURN also automatically establishes plug-in bindings from aspect markers to the aspectual properties of the concern. The UCM concept of stubs is therefore reused for aspect markers. The *AoView* for an aspect marker is a projection of the aspectual proper-

ties of the concern with those elements highlighted that are applicable to the aspect marker.

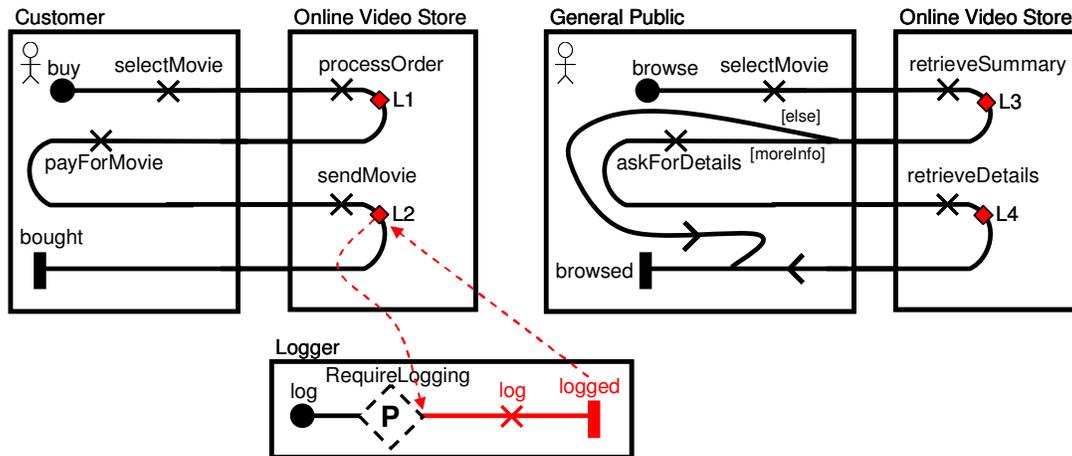


Figure 43 AoView of Logging Concern Applied to OVS System (Scenario)

5.3. Logging Concern (Goals)

While the previous sections have focused on AoURN scenario models, the Logging concern also impacts AoURN goal models. Figure 44 shows the goal models of three stakeholders of the OVS system. The Customer wants to own cheap movies and considers as options either borrowing from the library, buying at a mall store, or buying online. The General Public wants access to movie information which could be provided by movie magazines or online. The Online Video Store, on the other hand, wants to increase sales by improving the shopping experience and by improving its advertising strategy. The goal models also indicate the impact of the two scenarios introduced earlier on the various goals by modeling the Order Movie and Browse Movie scenarios as tasks. Last but not least, two dependencies are shown as both, the Customer and the General Public, depend on the Improve shopping experience goal of the OVS actor for buying online and using online information, respectively.

Similarly to the scenario model for the Logging concern, the goal model of the Logging concern also requires the specification of a) the aspectual properties of the Logging concern, b) the pattern that describes where the concern is to be applied, and c) the composition rule that states how the concern is to be applied. AoURN structures the goal models of concerns in a slightly different way than the scenario models of concerns, ow-

ing to the different nature of goal models as compared to scenario models because elements in goal models are generally much more connected than elements in scenario models. Furthermore, AoURN employs a tagging-based approach for the specification of concern elements in goal models.

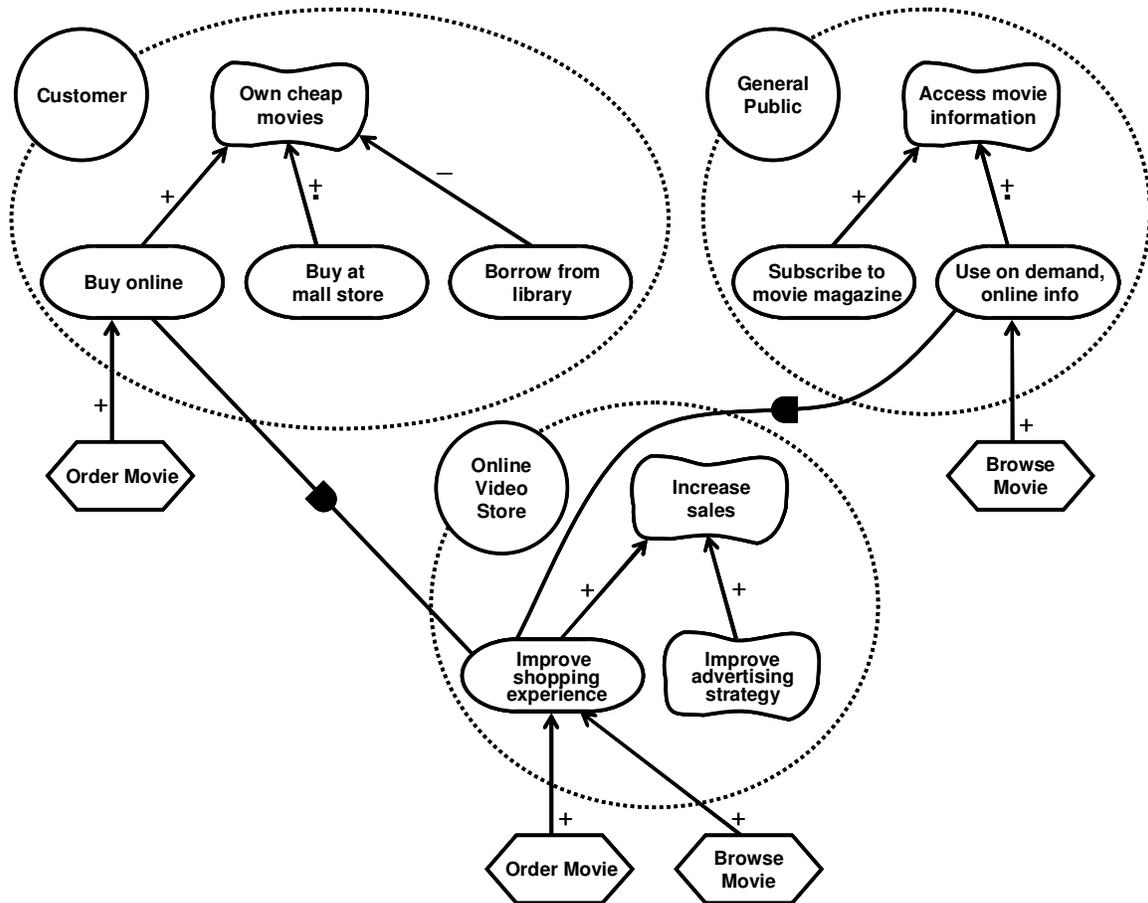


Figure 44 Goal Models of the Online Video Store System

The first modeling step again addresses the aspectual properties of the Logging concern as shown in Figure 45.a, introducing the Logging task and stating the softgoals for streamlining store operations and understanding user behavior. A URN link (▶) establishes traceability between the Logging task and the scenario model of the Logging concern (Figure 41.b). The second step, however, defines the pattern to be matched (Figure 45.b) while the third specifies the composition rule (Figure 45.c), reversing the two steps from the approach for scenario models.

The second step specifies the pattern, defining it as the Improve shopping experience goal and the Improve advertising strategy softgoal inside the Online Video Store

actor. The pattern is identified by tagging goal model elements with *pointcut stubs* (\diamond , Figure 45.b).

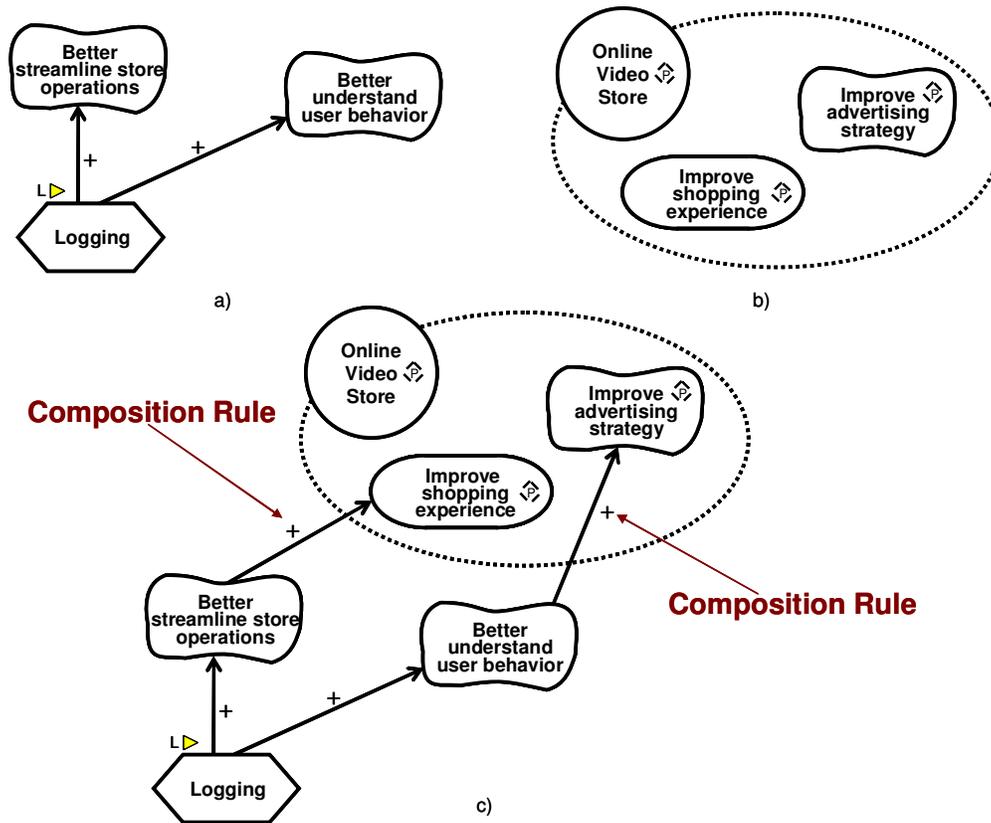


Figure 45 Step by Step Modeling of the Logging Concern (Goals)

The third step adds the composition rules to the goal model, by linking tagged elements from Figure 45.b with elements that are not tagged from Figure 45.a. The composition rule for the Logging concern in Figure 45.c therefore states that better streamlining of store operations positively contributes to improving the shopping experience and that a better understanding of user behavior also positively contributes to improving the advertising strategy. Clearly, the composition rule cannot be defined before the pattern for the goal models of a concern. The GRL graph in Figure 45.c is called the *pointcut graph*. The goal model for the Logging concern is now fully specified.

The impact of the Logging concern on the goal model is again indicated by tagging appropriate goal model elements automatically with *aspect markers* (\blacklozenge , Figure 46). Since the two intentional elements tagged with pointcut stubs inside the OVS actor, match two of the three intentional elements inside the OVS actor in Figure 44, aspect

markers for the Logging concern are added to the Improve shopping experience goal and the Improve advertising strategy softgoal. The addition of several aspect markers with the same name indicates that one match of the Logging concern impacts several model elements of the OVS goal model. Note that all dependencies are omitted from Figure 46.

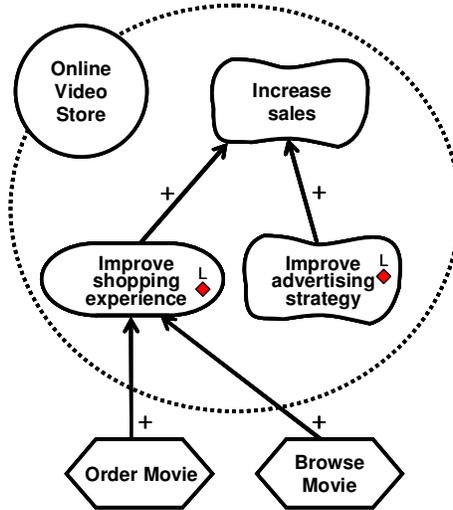


Figure 46 Impact of Logging Concern on OVS System (Goals)

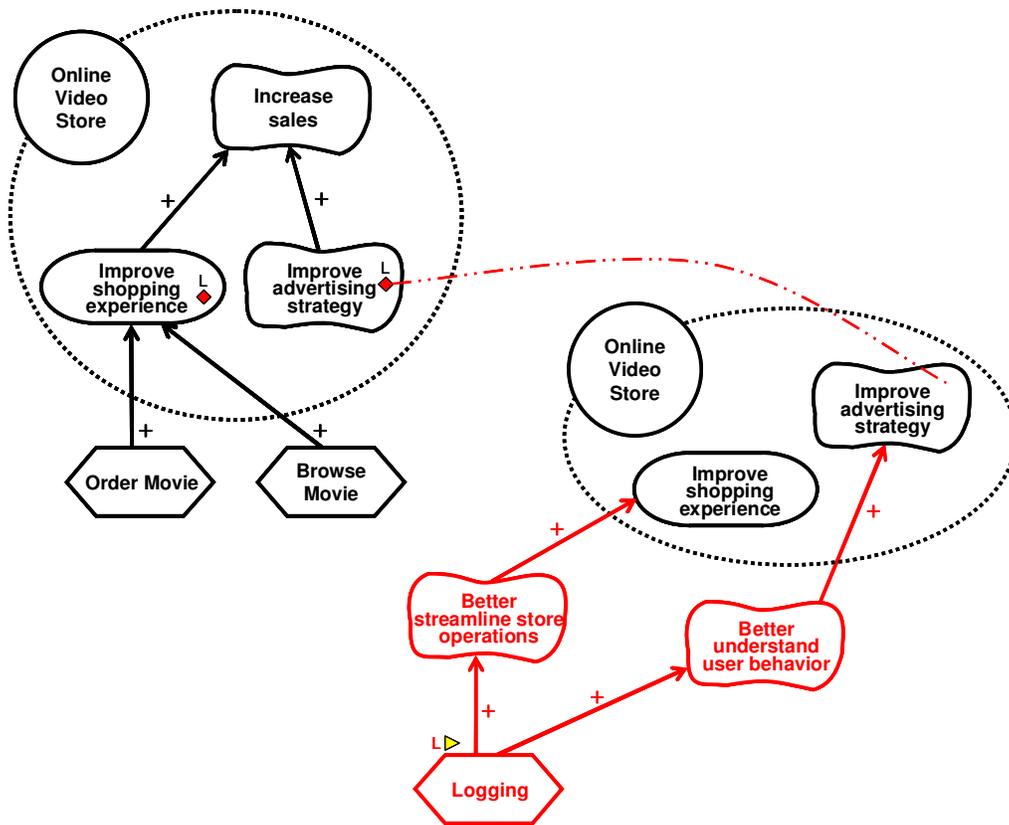


Figure 47 Detailed View of Logging Concern Applied to OVS System (Goals)

Further details about the impact of the Logging concern are provided again by the *AoView* which is shown for one of the aspect markers in Figure 47. The portion of the pointcut graph that applies to the aspect marker is highlighted, i.e., all composition rules and all aspectual properties from Figure 45.a that are connected to the model element mapped to the aspect marker. Hence, the *AoView* for goal models is a projection of the pointcut graph, similar to the *AoView* for scenario models being a projection of the aspectual properties. AoURN automatically manages the mappings between the aspect marker tags and the pointcut graph which allows for the *AoView* to be generated.

5.4. Authentication Concern (Scenario)

The second crosscutting concern to be added to the OVS system is the Authentication concern. As with the Logging concern, the same three steps yield the complete specification of the scenario model of the Authentication concern (Figure 48.a). The aspectual properties describe the Authentication scenario which starts at the authenticate start point and ends either at the authenticated or fail end point. Various conditions are checked along the way, the Customer may have to enter credentials, and the Security Server may authenticate or block the customer.

The composition rule for the Authentication concern requires the concern to be applied before the scenario requiring authentication, because the scenario is allowed to continue only if authentication is successful. Therefore, the pointcut stub is added to the model, so that the aspectual properties occur before the pointcut stub, thus indicating a “before” composition rule. Furthermore, the success path and not the fail path is connected to the pointcut stub, giving a first indication that only in the success case will the scenario be allowed to continue. The second indication is given by the usage of a local end point for fail. The semantics of a local end point does not allow such an end point to be further connected, thus indicating that the scenario stops with it.

Finally, the pattern is defined on the pointcut map. Since all OVS system actions caused by interactions of the Customer with the OVS system require authentication, the pointcut map matches any responsibility of the OVS that is on a path crossing from the Customer into the OVS. This pointcut map also means that the interactions of the General Public with the OVS system do not require authentication.

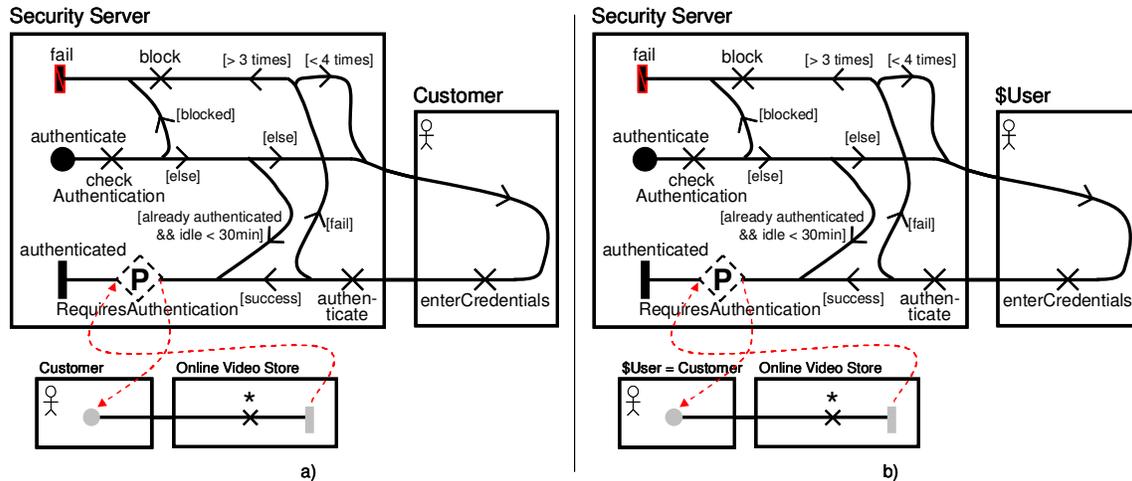


Figure 48 Initial and Improved Model of the Authentication Concern (Scenarios)

There is one major problem with the initial model in Figure 48.a. The aspect is very specific to the OVS system, because the Customer component is used directly in the aspectual properties. The aspect in Figure 48.a is therefore not as reusable as a more generic aspect, and that is obviously not ideal. The Customer component is used in the aspectual properties, because that is the component matched by the pointcut map. To make the aspect more generic, it is necessary to allow the aspect to reuse a component matched by the pointcut map. Therefore, the aspectual properties use a *variable* (\$) which references the pointcut map to make the aspect more generic and thus applicable to many systems. On the pointcut map, the variable is defined according to the context – in this case, the OVS system, i.e., \$User = Customer (Figure 48.b). If the aspect is applied to another system, the variable in the pointcut map can be adjusted easily while the aspectual properties do not have to be changed.

The impact of the Authentication concern is again indicated with aspect markers. The pattern on the pointcut map matches against the two responsibilities of the OVS component in the Order Movie scenario in Figure 40 that are on a path crossing from the Customer into the OVS. Two aspect markers for the Authentication concern are therefore added to this scenario before the matched pattern (Figure 49).

However, there is a fundamental difference between the Logging concern and the Authentication concern which should be and is reflected in the aspect markers. The Logging concern simply inserts aspectual behavior into the model without further changing

the existing model. The Authentication concern, on the other hand, may stop the scenario from continuing (as clearly defined by the local end point in Figure 48). It is important for the requirements engineer to immediately be able to differentiate these two cases. Therefore, *conditional aspect markers* (◆) are used in Figure 49 instead of regular aspect markers. Conditional aspect markers indicate that the scenario may not continue past the aspect marker, whereas regular aspect markers guarantee that the scenario continues past them. Figure 49 shows the AoView for one of the aspect markers. If a variable is used by the aspect, the AoView shows the actual match and not just the variable. Therefore, Customer is shown in Figure 49 instead of \$User.

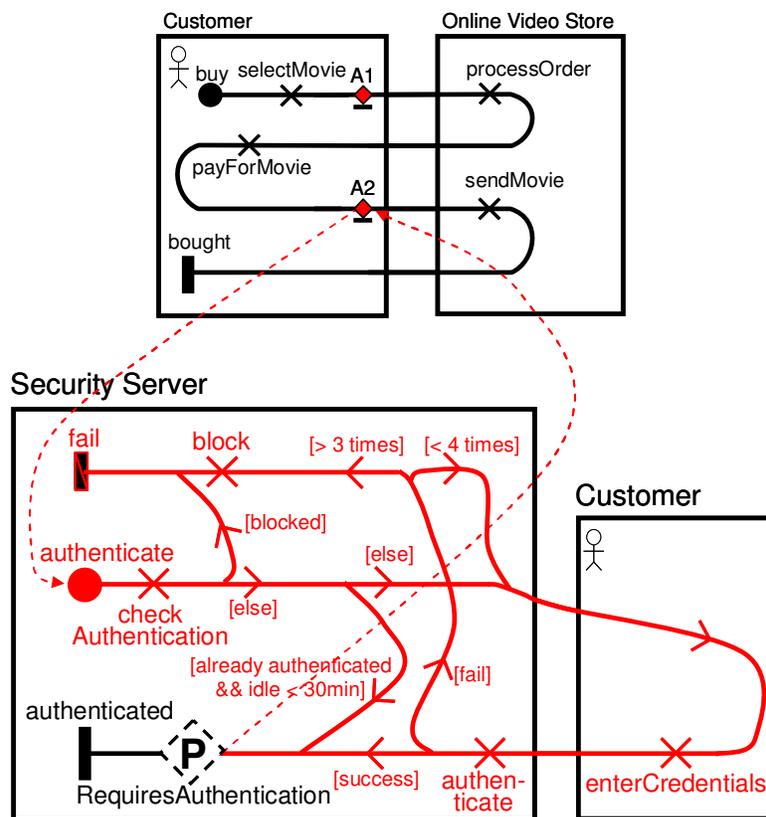


Figure 49 AoView of Authentication Concern Applied to OVS System (Scenario)

5.5. Authentication Concern (Goals)

The goal model for the Authentication concern is more complex than the one for the Logging concern, but the same three modeling steps still apply. First, the aspectual properties are modeled. Authentication is part of the large Security concern which warrants a more

for both aspect markers. If the aspect uses a parameterized expression in the pointcut graph, the AoView shows the actual match and not just the parameterized expression. Therefore, Online Video Store and Improve shopping experience are shown in Figure 51 instead of * for the bottom right aspect marker, i.e., the match of the Order Movie task connected to OVS. For the other aspect marker, i.e., the match of the Order Movie task connected to Customer, the AoView shows Customer and Buy online instead of *.

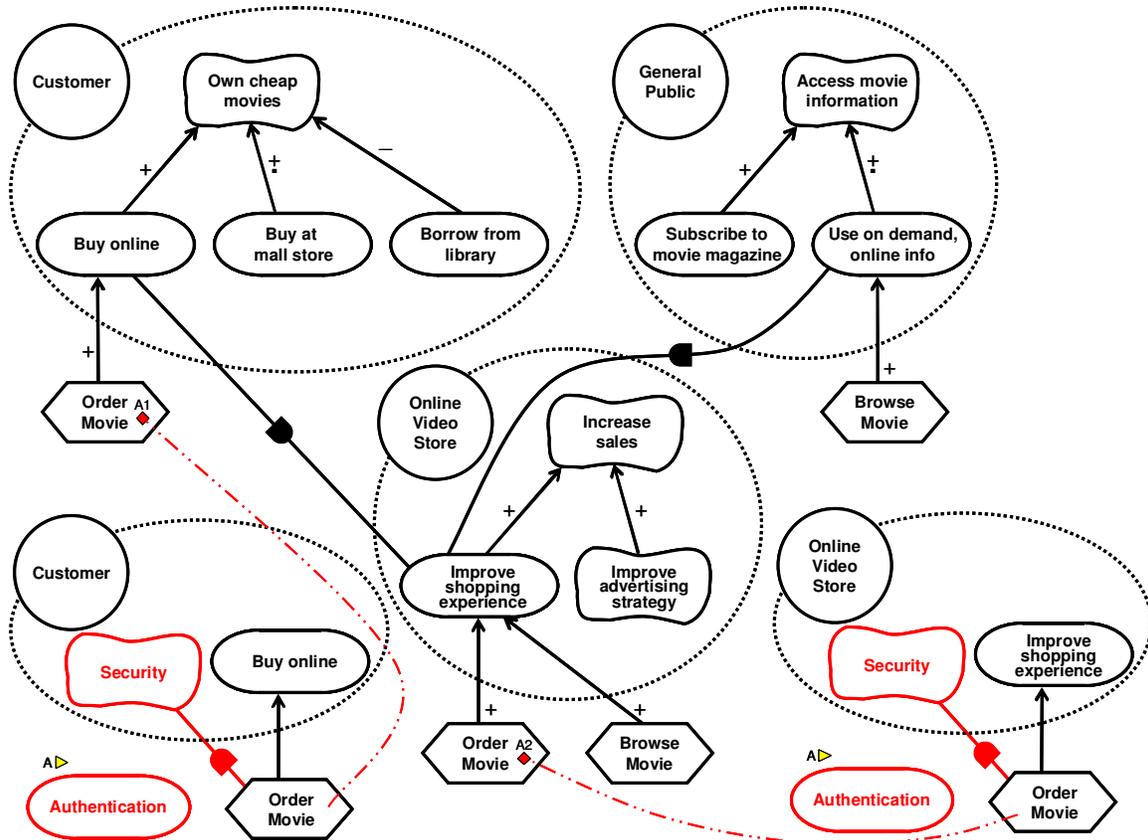


Figure 51 AoViews of Authentication Concern Applied to OVS System (Goals)

5.6. Communication Concern

The Communication concern is quite different from the Logging and Authentication concerns as it is a rather low-level design concern. The UCM notation abstracts from communication and message details and simply shows a path crossing from one component into another. At later stages in the development process, however, it becomes more important to define these details. The Communication concern does that and demonstrates

that aspects that inherently exist in UCM may be added with AoURN without complicating the original model.

The intent of the Communication concern is illustrated with model excerpts in Figure 52. Instead of abstracting away from any particular communications protocol between the Customer and the OVS, the details of the request/reply communication pattern are modeled. The abstract version in Figure 52.a shows an interaction of the Customer with the OVS as an action of the Customer, a path crossing from the Customer component into the OVS system, a response action of the system, and a path crossing back into the Customer component. The more detailed version in Figure 52.b shows explicit request and reply responsibilities and indicates that the Customer waits until a reply is received. Furthermore, if a reply is corrupted, then the interaction fails and may be retried again.

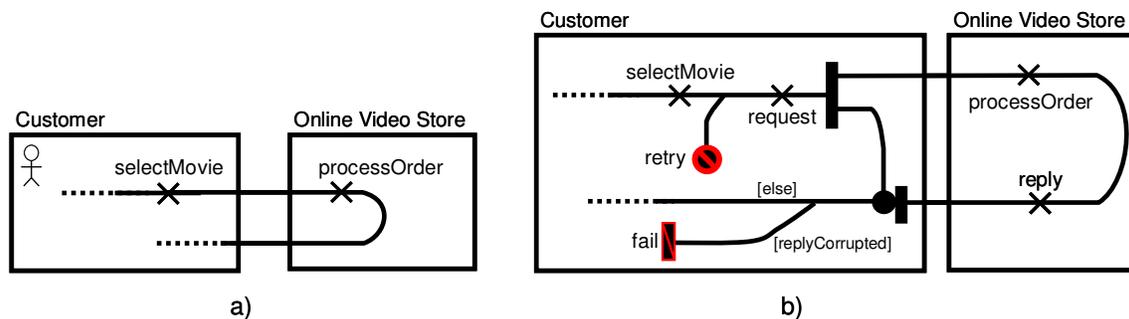


Figure 52 Intent of Communication Concern

The change imposed by the Communication concern on the abstract version is quite complicated. It is actually easier to fully replace the abstract version with the more detailed version instead of trying to insert behavior only at the right locations as is done for the Logging and Authentication concerns. The *replacement pointcut stub* (⊗, Figure 53) indicates just that, i.e., the pattern matched by the pointcut map is removed from the model and fully replaced by the aspectual behavior of the aspect. By convention, the replacement pointcut stub is always shown right after the start point.

As the Communication concern is to be applied to many interactions in the model, it is obvious that variables have to be used by the aspect to describe its aspectual properties in a generic way. The Customer, selectMovie, the Online Video Store, and processOrder in Figure 52.b may change if the Communication concern is applied to a different

system. Therefore, they are replaced by the variables \$Requester, \$initiateRequest, \$Replier, and \$performRequest, respectively.

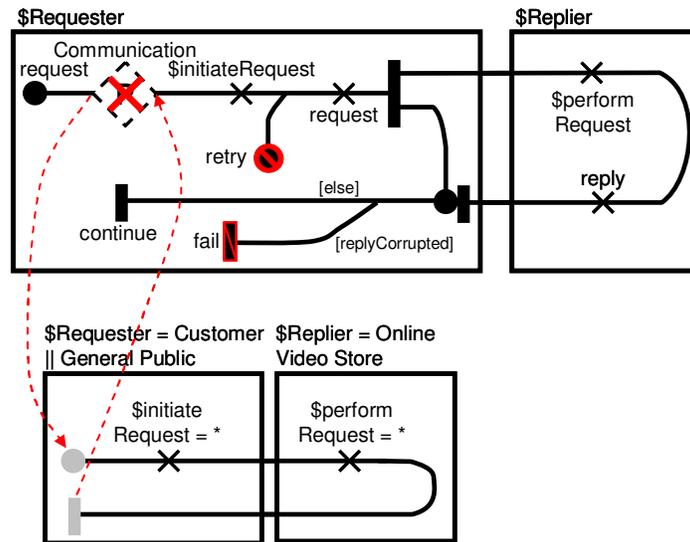


Figure 53 Modeling the Communication Concern

Note also that the local start point `retry` in Figure 53 indicates that it is available only when the Communication concern is active, while the local end point `fail` in Figure 53 differentiates the fail end point from the continue end point. The scenario is allowed to only continue past the `continue` end point while the fail end point stops the scenario.

Finally, the pattern described by the pointcut map matches any path going from the Customer to the Online Video Store and back with any responsibility in each component. It also defines the four variables used by the aspect in the context of the OVS system. Figure 53 thus presents the full specification of the scenario model for the Communication concern.

The impact of the Communication concern on the model is as usual visualized by the presence of aspect markers. However, the fact that the Communication concern performs a replacement and not an insertion influences the way the aspect markers are used. Instead of a regular aspect marker being added to the model for each insertion of aspectual properties, a pair of aspect markers called *tunnel entrance aspect marker* (\blacklozenge) and *tunnel exit aspect marker* (\blacklozenge) is used. As the pattern of the Communication concern is matched four times in Figure 40, Figure 54 shows four pairs of aspect markers. Each pair is identified by a number.

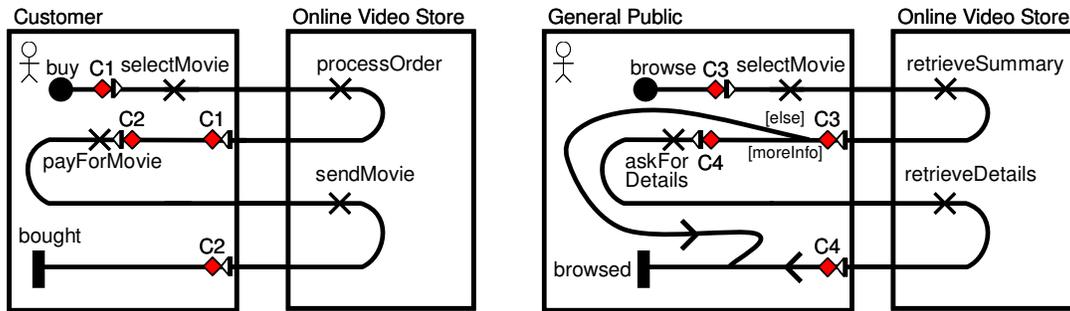


Figure 54 Impact of Communication Concern on OVS System

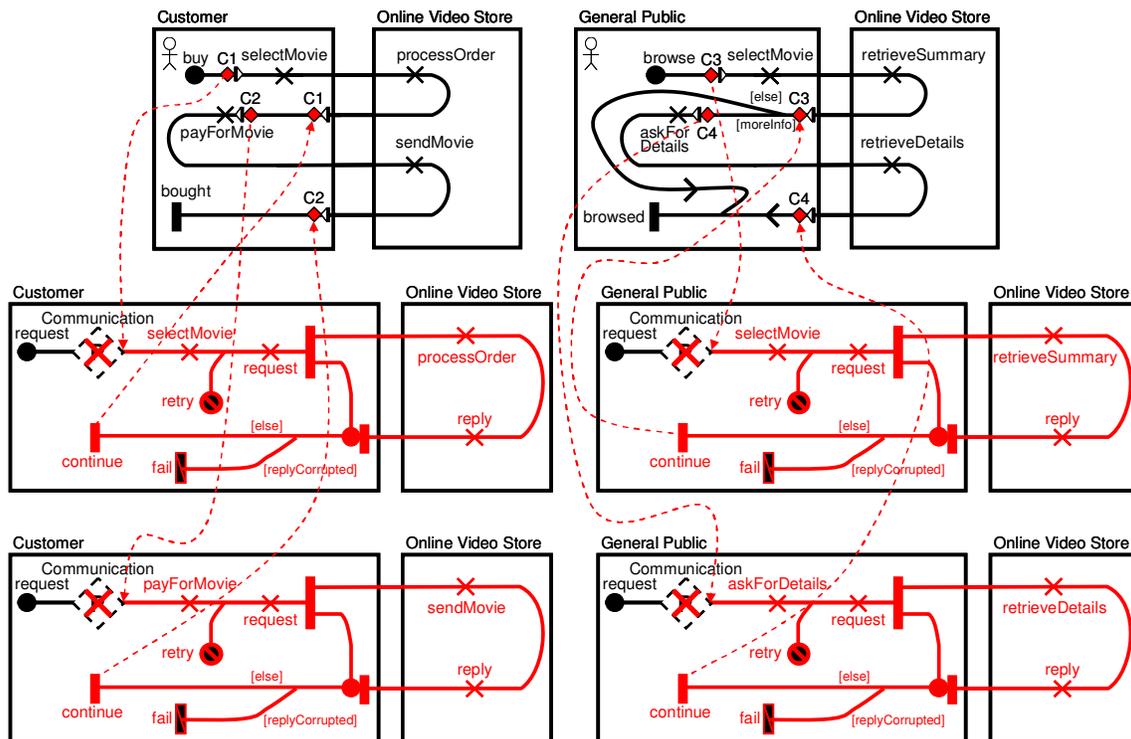


Figure 55 AoViews of Communication Concern Applied to OVS System

While regular aspect markers have plug-in bindings to and from the map where the aspectual properties are defined (see e.g., Figure 43), tunnel aspect markers have only one plug-in binding. More specifically, the tunnel entrance aspect marker has a plug-in binding to a map, whereas the tunnel exit aspect marker has a plug-in binding from a map. Consequently, the tunnel aspect markers work together to circumvent the original behavior, effectively replacing it with the aspectual behavior. As with regular aspect markers, AoURN automatically inserts tunnel aspect markers and automatically establishes their plug-in bindings. The AoViews for the four pairs of aspect markers in Figure 55 shows

Aspect markers for the Logging concern appear in the Communication concern, because the pointcut expression of the Logging concern also applies to the responsibilities of the OVS in the Communication concern, ensuring that all responsibilities of the OVS are still logged. Similarly, aspect markers for the Authentication concern appear in the Communication concern, because the pointcut expression of the Authentication concern also matches against the Communication concern, making sure that all customer interactions with the OVS are still authenticated. Because both pointcut expressions apply to the Communication concern, it does not matter that the Communication concern effectively removes four path segments from the Order Movies and Browse Movies scenarios. For example, the tunnel aspect markers C1 remove `selectMovie`, the Authentication aspect marker, `processOrder`, and the Logging aspect marker in the top left map, but these two aspect markers also appear in the middle left map.

Consider, however, an Encryption concern that guarantees that any customer interaction with the OVS is properly encrypted and decrypted. Its pointcut expression matches a path going from the Customer component to the OVS component and therefore inserts behavior at the same location as the Authentication concern. The desired composition of these two concerns is shown in Figure 57. The first aspect marker for the Encryption concern encrypts the interaction, while the second decrypts the interaction. In this case, there is a conflict between the Authentication concern and the Encryption concern. The Authentication concern must be applied before the Encryption concern, because encrypting something that will not be used because authentication fails is a waste of resources.

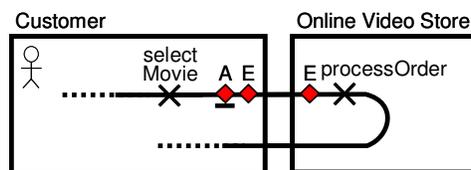


Figure 57 Interaction of Encryption and Authentication Concerns

To properly deal with concern interactions, dependencies ($\text{---}\blacksquare\text{---}$) and conflicts ($\text{---}\rightarrow$) and their resolutions are modeled in the *concern interaction graph* (CIG) which is identified by the tag `<<interaction>>`. Resolutions are modeled in the CIG by the direction of links. A target of a link must be applied before the source of a link. The CIG in Figure 58 states

that the Authentication concern conflicts with the Encryption concern and must be applied before it. Dependencies do not exist for the OVS example.



Figure 58 Concern Interaction Graph for OVS System

Undesired concern interactions are not constrained to just AoUCM models, but they may also manifest themselves in AoGRL models. For example, two concerns may add contradictory links between the same intentional elements.

5.8. Summary

This chapter gives an intuitive overview of the Aspect-oriented User Requirements Notation (AoURN) with the help of the Online Video Store example to which three concerns, Logging, Authentication, and Communication, are applied. The interactions among these concerns are also discussed. AoURN's details are presented in the following chapter.

Chapter 6. Aspect-Oriented User Requirements Notation

This chapter first gives an overview of the Aspect-oriented User Requirements Notation (AoURN) and then introduces in detail Aspect-oriented GRL (AoGRL) and Aspect-oriented UCM (AoUCM). The features of AoURN presented in this chapter allow the specification of crosscutting concerns while ensuring concern encapsulation regardless of whether the concerns are crosscutting or not. Furthermore, the navigation of AoURN models in an aspect-oriented way without the need to explicitly compose AoURN models into standard URN models is discussed and illustrated. Advanced features of AoURN are then discussed for the specification of more complex crosscutting concerns. AoURN is initially presented in an informal way and towards the end of this chapter more formally with the AoURN metamodel. The static semantics of all concepts is defined precisely in the AoURN metamodel, while the dynamic semantics of AoUCM is defined with the behavior of the UCM traversal mechanism and the GRL evaluation mechanism. The chapter concludes with a discussion of the relationship of concerns in goal and scenario models. Throughout the chapter, several examples demonstrate the use of AoURN in requirements modeling.

6.1. Overview

The Aspect-oriented User Requirements Notation (AoURN) [86][91][95][98][103] unifies goal, scenario, and aspect-oriented modeling in one framework. AoURN allows requirements engineers to better encapsulate requirements concerns which are hard or impossible to encapsulate with URN models. In the context of URN, the major concerns are use cases and non-functional requirements, all of which can be modeled by aspects with AoURN. In addition, all goals specific to a stakeholder are also modeled as a concern with AoGRL. As URN has been used in the context of various software development paradigms (see Section 2.1 for examples), URN with aspect-oriented extensions is also

applicable to a wide variety of development paradigms. Over the last years, AoURN has been successfully applied in the context of safety-critical systems [95], feature interactions [93][97][100][101], business process management systems [120][121], software product lines [94], SOA-based systems [27], and e-commerce systems [91][92][98].

To fully support aspect-oriented modeling with AoURN, the basic aspect concepts mentioned in Section 2.2 have to be introduced to the User Requirements Notation (URN). These concepts are the join point model, aspectual properties, pointcut expressions, composition rules, and concerns.

6.1.1 Join Point Model

AoURN first extends URN by defining a *join point model* for the Goal-oriented Requirements Language (GRL) and Use Case Maps (UCM). The join point model specifies the potential locations in the URN model, i.e., the *join points*, which may be modified by an aspect.

All nodes and links of GRL graphs and all UCM path elements optionally residing within the boundary of an actor or component, respectively, are deemed to be join points, except for purely visual elements such as direction arrows. By including all modeling elements with semantic meaning in the join point model, the requirements engineer is given the power to change any part of a URN model with aspect-oriented techniques.

6.1.2 Aspectual Properties

The properties of an aspect comprise goals, behavior, and structure in the case of URN. AoURN allows *aspectual properties* to be defined on standard URN diagrams called *aspect diagrams* (*aspect graphs* for AoGRL and *aspect maps* for AoUCM). This approach ensures that the requirements engineer can continue working with familiar models.

6.1.3 Pointcut Expressions

Pointcut expressions describe a pattern which defines where in the base model an aspect is applied by matching join points against the pointcut expressions. Each match establishes *mappings* between the pointcut expression and join points in the base model. Since the join point model contains all significant URN modeling elements, pointcut expres-

sions for AoURN models may identify any URN model element. In addition, a sequence of URN model elements may also be identified, enabling the requirements engineer to impact a larger portion of a URN model with aspects. AoURN pointcut expressions must have a behavioral or intentional dimension and may additionally have a structural dimension.

Pointcut expressions are defined visually on *pointcut diagrams* (*pointcut graphs* for AoGRL and *pointcut maps* for AoUCM) that are matched against the rest of the model. Pointcut diagrams are standard URN diagrams, allowing the requirements engineer to again continue working with familiar models. Pointcut diagrams may also be parameterized for increased matching power by allowing names of modeling elements to contain wildcards (*) and logical expressions (containing “and”, “or”, “xor”, and “not”).

In the case of AoUCM, aspect maps are loosely coupled to pointcut maps, allowing aspect maps and pointcut maps to be reused independently from each other. In the case of AoGRL, pointcut graphs do not exclusively model pointcut expressions. Instead, aspectual properties and pointcut expressions are more intertwined with each other owing to the interdependent nature of GRL graphs.

6.1.4 Composition Rules

Once the aspectual properties and the pointcut expressions of an aspect are defined, the base model may be transformed by the aspect as defined by *composition rules*. Composition rules state how the aspectual properties are to be applied to the join points matched by the pointcut expressions. Typically, standard composition rules allow aspects to be applied before, after, or instead of the join point.

AoURN composition rules are much more flexible because they are defined with URN itself, and are therefore as expressive as URN and not restricted by the capabilities of any particular pointcut language (which, for example, often allow only standard composition rules). Join points matched by a pointcut expression and transformed by a composition rule are automatically indicated with small, filled diamonds called *aspect markers* (◆), identifying the insertion points for aspectual properties in the base model. Often, an aspect marker is labelled with an abbreviation for the concern associated with the aspect marker. Any AoURN tool must establish and retain the mappings and aspect mark-

ers in order to navigate and reason about the AoURN model in an aspect-oriented way. For example, double-clicking on an aspect marker may present a list of all applied aspects to the requirements engineer. Selecting one of them then takes the requirements engineer to an *AoView*, i.e., a visualization of the aspectual properties inserted at the location of the aspect marker.

6.1.5 Concerns

A *concern* is simply an organizational construct that contains all URN diagrams required to describe a unit of interest to a requirements engineer, e.g., a use case, feature, non-functional requirement, or stakeholder. In the case of an *aspect* (i.e., a crosscutting concern), it contains (a) any number of aspect diagrams, (b) any number of pointcut diagrams, and (c) any number of regular URN diagrams required to describe the crosscutting behavior of the aspect. In the case of a non-crosscutting concern, the concern contains only regular URN diagrams as required to describe the concern.

A well-documented fact of aspect-oriented software development is that concerns interact with each other in possibly undesirable ways. AoURN addresses this issue with the *concern interaction graph* (CIG) [88], which documents conflicts and dependencies between concerns. Simple syntactical conflicts between concerns may arise when they are applied at the same join point in the model. Dependencies exist when a concern can be applied only if another concern has been applied before because, for example, model elements of the other concern and base elements are together required for a successful match of the concern's pointcut expression. Concern *precedence rules* establish an order between conflicting or dependent concerns that determines which concern is applied first. More complicated, semantic conflicts that cannot be resolved by precedence rules may also occur between concerns [100][101][102], possibly requiring substantial changes to the concerns themselves.

Figure 59 gives an abstract example of a CIG, showing typical stakeholder (○, e.g., Stakeholder Concern 1), use case (◁, e.g., Use Case Aspect 1), and non-functional requirements (◻, e.g., Non-functional Aspect 1) concerns and their relationships (—●—, ⋯→).

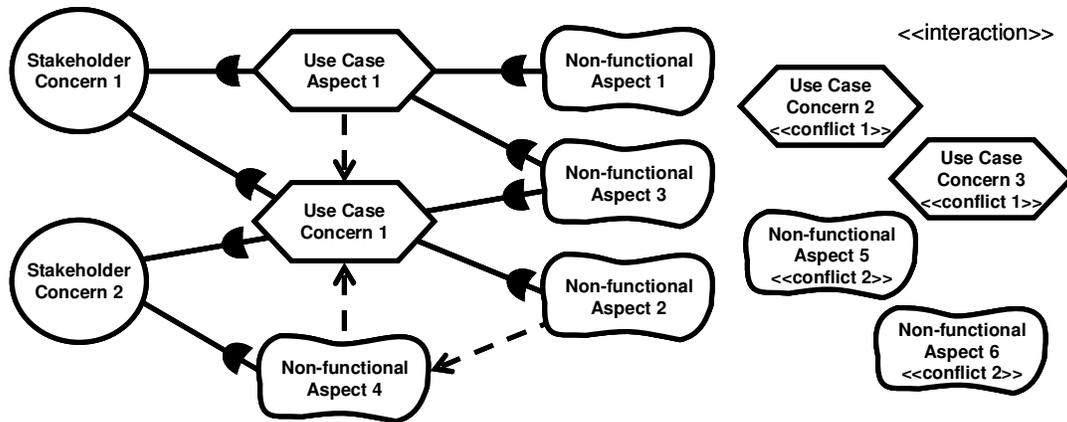


Figure 59 Concern Interaction Graph

As it is not always possible to immediately resolve conflicts between concerns, the CIG allows the requirements engineer to indicate that a non-resolved conflict exists between two or more concerns with the help of the metadata name/value pairs *aspect/conflict N* (e.g., <<conflict 1>> for Use Case Concern 2 and Use Case Concern 3). This approach is also useful for concerns with semantic conflicts or cyclical dependencies that require more wide-reaching changes to the model. These concerns can be documented as conflicting while the changes are made. Any intentional element on the CIG may be tagged with the *aspect/conflict N* metadata. Concerns that are not shown on the CIG are deemed to not have any interactions with any other concern.

Dependency links are used to show that a concern depends on another (e.g., Use Case Aspect 1 depends on Stakeholder Concern 1 to be applied before it can be applied itself). Correlation links are used to indicate conflicts and their resolutions (e.g., Use Case Aspect 1 and Use Case Concern 1 are applied at the same join point in the model and therefore conflict with each other). The direction of a correlation link shows the resolution of the conflict. In this case, the concern precedence rule establishes that Use Case Concern 1 has priority over Use Case Aspect 1, as Use Case Concern 1 is the target of the correlation link while Use Case Aspect 1 is the source. The concern interaction graph is a standard GRL graph that is distinguished from other standard GRL graphs with the help of the metadata name/value pair *aspect/interaction*.

Since aspect diagrams are structurally the same as standard URN diagrams, it is possible to define aspects on other aspects. By default, pointcut expressions are also matched against aspectual properties in addition to standard URN diagrams. The match-

ing process, however, does not assume by default that an aspect is applied to the model. Therefore, a pointcut expression of aspect A that matches partially against an aspect diagram of aspect B and partially against another URN diagram results only in a successful mapping of the pointcut expression to the aspect and the other URN diagram, if a dependency relationship from aspect A to aspect B is defined in the concern interaction graph. Only in this case, aspect B is applied to the model first before an attempt to match the pointcut expression of aspect A is made. After application of aspect B, the complete pattern matched by the pointcut expression of aspect A exists in the model and is therefore successfully matched.

Note that dependencies are not required if the pointcut expression of one aspect directly matches another aspect without the involvement of any other URN diagrams. While it is true that this is also some form of dependency, it is also rather fundamental behavior of aspect-oriented systems that does not need to be explicitly specified. Otherwise, all matches would have to be modeled in the concern interaction graph. This also means that two concerns can be applied to each other at the same time.

Extensions to the URN Metamodel – The extensions to the URN metamodel required to support the concern concept are rather small (Figure 60). A Concern is itself a new URNmodelElement that may group any number of URNmodelElements. For AoURN, this grouping is restricted to URN diagrams, other concerns, and GRL intentional elements on the CIG. By assigning an element on the CIG to a Concern, the concern concept can be characterized by its type. Furthermore, a Concern has a Condition which defines whether the concern is to be applied, i.e., the expression of the Condition evaluates to true, or whether the concern is not to be applied, i.e., the expression of the Condition evaluates to false. As a Concern is a URNmodelElement it has a name attribute. In addition, a Concern also has an abbreviation attribute which is used for labelling aspect markers associated with the Concern.

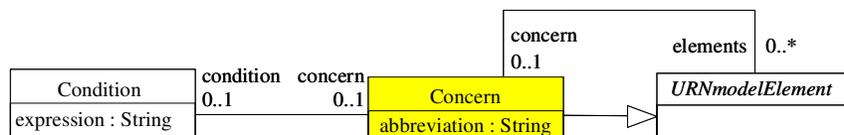


Figure 60 URN Metamodel Extensions: Concern

6.2. Aspect-oriented GRL

Aspect-oriented GRL (AoGRL) [86][95][96][103] adds support for early aspect-oriented modeling to GRL. The following sub-sections describe in more detail the relevant basic aspect concepts listed in Section 6.1.

6.2.1 Join Point Model

The *join point model* for GRL is shown in Figure 61. Actors as well as all intentional elements in GRL (softgoals, goals, tasks, resources, and beliefs) and all links in GRL (contribution, correlation, decomposition, and dependency) optionally residing within the boundary of an actor are deemed to be *join points*. Associated with join points are *insertion points* that precisely indicate a location in the goal graph that may be transformed by an aspect. An actor or an intentional element has exactly one insertion point, while a GRL link has two insertion points – one for the source and one for the target of the link.

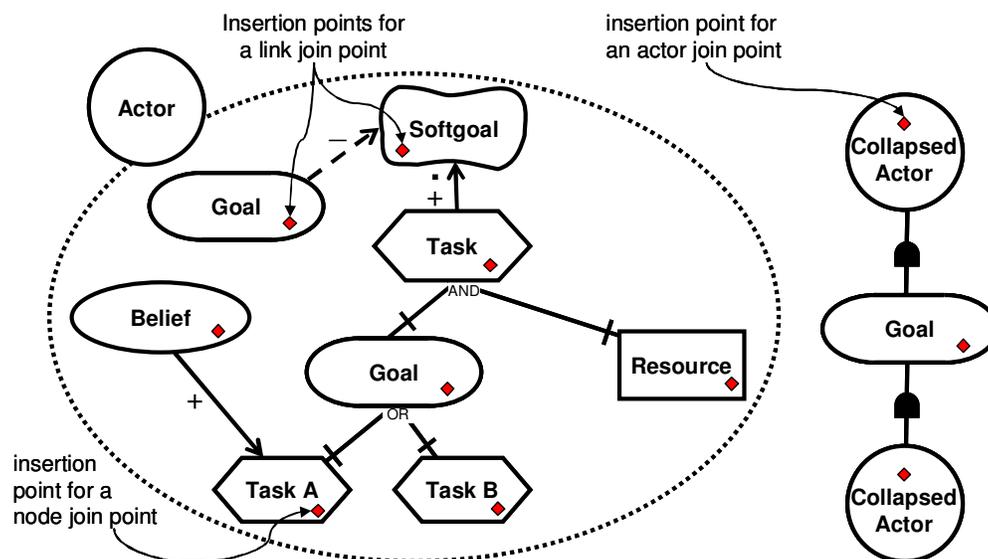


Figure 61 GRL Join Point Model

6.2.2 Aspectual Properties

The properties of an aspect, in this case intentional elements, links, and actors, are specified on a standard GRL graph called an *aspect graph* which is very similar to the notion of reusable GRL catalogues, if the catalogue describes the goal model of only one concern. However, traditional GRL models face difficulties when trying to reuse GRL cata-

logues several times within the same GRL model, leading to renaming and duplication of the content of the GRL catalogue. AoGRL adds the ability to include concern-specific GRL catalogues multiple times into a GRL model without duplication of modeling elements by specifying pointcut expressions that match all desired join points in the GRL model. An AoURN tool understands that an individual instance is added each time elements of the aspect graph are added to the goal model. Figure 62 gives an example of an aspect graph that contains a simple goal tree with three intentional elements.

6.2.3 Pointcut Expressions

GRL pointcut expressions are visually specified on a *pointcut graph*. All actors, nodes, and links in a pointcut expression are identified by *pointcut markers* (\oplus) or *pointcut deletion marker* (\otimes). For example, the pointcut graph in Figure 62 matches against all goal graphs that contain a goal Stakeholder Goal C1 which is an OR decomposition of a softgoal ending with Softgoal B and has a correlation with another softgoal, all of which have to reside within an actor. The long-dash-dot-dotted lines without arrowheads illustrate the mappings for the join points in the base model matched by the pointcut expression. Note that the ability to mark elements in a pointcut expression is the only extension required for a URN tool in order to specify AoGRL models. The marking is realized with the help of the metadata name/value pairs *aspect/pointcut marker* and *aspect/pointcut deletion marker*. Any element on a pointcut graph may be tagged with these metadata name/value pairs. As the example shows, pointcut expressions may be parameterized. See Appendix C: BNF for Name Expressions for a more formal definition of names in pointcut expressions.

A pointcut graph contains not just a pointcut expression, but also other intentional elements not identified with the pointcut marker. These unmarked, intentional elements are part of the aspectual properties and are inserted at the join points matched by the pointcut expression. They either reference elements with the same name in the aspect graph or are new pointcut-specific elements introduced by the aspect (Task of Aspect and Pointcut Specific Softgoal, respectively, Figure 62).

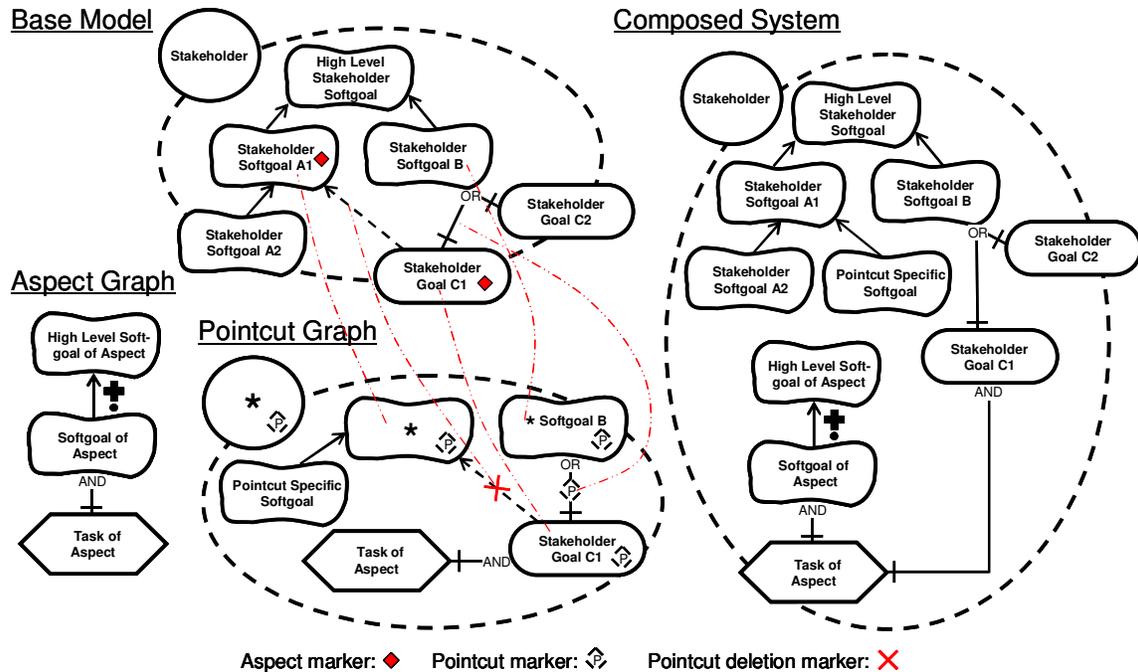


Figure 62 Basic Elements of AoGRL Notation

6.2.4 Composition Rules

In AoGRL, the *composition rule* for an aspect is defined by the set of unmarked links connecting unmarked nodes with elements belonging to the pointcut expression on the pointcut graph. As a result, the unmarked elements are added to the GRL model including any disjoint elements. The composition rule may also state that matched elements should be removed with the help of the *pointcut deletion marker* (✗). For example, the composition rule in Figure 62 stipulates that Task of Aspect and Pointcut Specific Softgoal are to be connected to Stakeholder Goal C1 with a decomposition and the matched softgoal with a contribution, respectively. Furthermore, the correlation between Stakeholder Goal C1 and the matched softgoal is to be removed. Hence, if an existing actor is specified by the pointcut expression, GRL elements can be added to or removed from it with AoGRL. If a GRL element is added to an actor by the aspect but already exists for the actor, then the added GRL element is merged with the existing element of the actor in the GRL model.

Composition rules are applied to each join point matched by the pointcut expression. Composition rules for AoGRL are exhaustive in that they cover all possible rela-

tionships that can be expressed with GRL (i.e., they are not limited by a particular pointcut language to a subset of the relationships).

The matched join points are identified by aspect markers in the combined model, as already stated in Section 6.1.4. *Aspect markers* (◆) are small, filled diamonds that indicate the insertion points in the base model and are added automatically to the AoGRL model by AoURN tools. Aspect markers are realized with the help of the metadata name/value pair *aspect/aspect marker*. Any GRL intentional element may be tagged with the *aspect/aspect marker* metadata. For example in Figure 62, new properties are added to or removed from the base model at the softgoal Stakeholder Softgoal A1 and the goal Stakeholder Goal C1, and aspect markers are therefore shown for both GRL nodes. Stakeholder Softgoal B, on the other hand, does not have an aspect marker because the softgoal is not transformed by the aspect even though the softgoal is matched by the pointcut expression and a mapping from the softgoal to the pointcut graph exists. Aspect markers that are inserted because of the same match of the pointcut expression (as is the case in Figure 62) refer to the same instance of the aspectual properties of the aspect. The result of applying the aspect to the base model is shown as a traditional GRL model in the composed system on the right side of Figure 62. Composition of AoGRL models is explained in more detail in Chapter 7.

Ideally, a pointcut graph should contain only the pointcut expression. An alternative to the above approach is therefore to separate the pointcut expressions and the unmarked elements of the pointcut graph into two separate GRL graphs. Nevertheless, the composition rule still needs to be shown visually to give an immediate indication of the impact of an aspect. This approach is generally the approach suggested by Alencar *et al.* [2]. However, given the highly interconnected nature of GRL models, this approach causes the unnecessary duplication of the composition rule. First, the visual element used to summarize the unmarked elements in the pointcut graph still needs to be connected to the pointcut expression. Second, the summarized visualization alone is not sufficient to specify the exact composition rule, therefore requiring a representation of the relationship of the visual element summarizing the unmarked elements and the unmarked elements themselves. Hence, this alternative is not further pursued for AoGRL.

6.3. Aspect-oriented UCM

Aspect-oriented Use Case Maps (AoUCM) [86][90][91][95][98][103] adds support for early aspect-oriented modeling to UCM. The following sub-sections describe in more detail the relevant basic aspect concepts listed in Section 6.1.

6.3.1 Join Point Model

The *join point model* for UCM is shown in Figure 63. In aspect-oriented programming (AOP), a join point is a point in the dynamic flow of the program such as a method call, the execution of a method, the initialization of an object, set methods, get methods, or exception handling. Hence, a join point has a behavioral dimension as well as a structural dimension. For use cases, a join point is defined as a step in a flow of the use case [61]. In UCM terms, this definition translates directly into responsibilities, i.e., the behavioral dimension, optionally bound to components, i.e., the structural dimension. Responsibilities, however, are just one kind of path node. Therefore, any path node could be a join point. Defining any path node as a join point a) gives the most flexibility to the requirements engineer without significantly increasing the complexity of AoUCM or requiring additional modeling constructs, and b) moves AoUCM closer to a general pattern matching and transformation approach for aspects. Therefore, all UCM path nodes optionally bound to components are deemed to be *join points* except for purely visual nodes such as direction arrows. This approach allows an aspect to add its behavior and structure not only relative to responsibilities (i.e., functionality) but also relative to structural information and to causal flow constructs such as alternatives, concurrency, and loops.

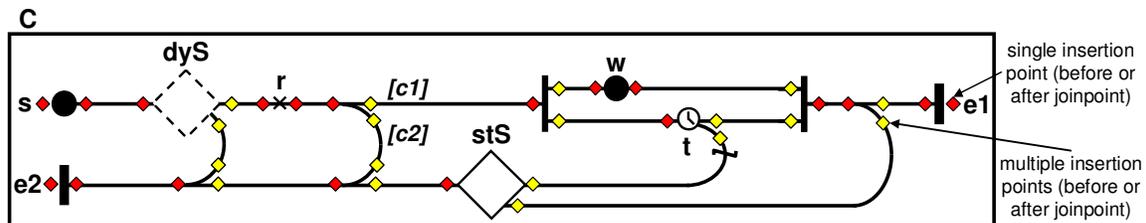


Figure 63 UCM Join Point Model

In Figure 63, the small diamonds do not indicate join points but *insertion points*, the precise location on a path that may be transformed by an aspect. An insertion point is associated with exactly one join point. All join points have at least two insertion points, one be-

fore and one after the join point. Some join points such as responsibilities and end points also have at the most two insertion points. Other join points such as stubs, forks, and joins may have more than two insertion points before or after the join point, as indicated by small light-shaded diamonds. The number of insertions points is directly related to the number of paths leaving or arriving at a join point. Each path between two join points therefore has exactly two insertion points, one after the source and one before the target. This establishes an ordering of these two insertion points – the one after the source occurs before the one before the target.

6.3.2 Aspectual Properties

The specification of the properties of an aspect, i.e., in this case behavior and structure, is straightforward with the standard UCM notation as UCM is meant to do exactly that: describe behavior with paths on top of a structure of components. Components may describe aspects just like any other structural entity as the semantic meaning of UCM components is cast very wide. Paths in an aspect component may reuse already existing responsibilities or may add new responsibilities. The properties of an aspect are therefore shown on a standard UCM map called an *aspect map*. For example in Figure 64.a, an aspect with a component C and three responsibilities Behavior.before, Behavior.after_success, and Behavior.after_fail is shown. Furthermore, since aspect maps may contain references to existing components, it is possible for an aspect to add behavioral or structural elements to existing components.

6.3.3 Pointcut Expressions

AoUCM extends UCM with the ability to specify pointcut stubs on aspect maps, thus enabling aspect-oriented modeling. *Pointcut stubs* (⊕, Figure 64.a) are structurally the same as dynamic stubs but have a slightly different semantic meaning. While dynamic stubs contain plug-in maps that further describe the structure and behavior of a system, pointcut stubs contain zero or more pointcut maps that visually describe pointcut expressions. Note that the ability to define pointcut stubs is one of two extensions required for URN tools to specify AoUCM models (for the other, see Section 6.5.2).

An aspect map differs from a traditional map (e.g., Base Model, Figure 64.b) only in that it contains one or more pointcut stubs. For example, the pointcut stub in Figure 64.a contains one *pointcut map* that matches against all maps that contain an OR-fork followed by a responsibility on at least one branch. Start and end points without labels on a pointcut map are not included in the match, but denote only the beginning and end of the pointcut expression to be matched. Therefore they are shown in grey in Figure 64.a. A pointcut map therefore represents a partial map which identifies join points when matched against other maps in the UCM model. The long-dash-dot-dotted lines without arrowheads in Figure 64.b illustrate the mappings for the join points in the base model matched by the pointcut expression. The OR-fork of the pointcut expression is matched against the OR-fork in the base model and the responsibility * in the pointcut expression is matched against responsibility R1 in the base model.

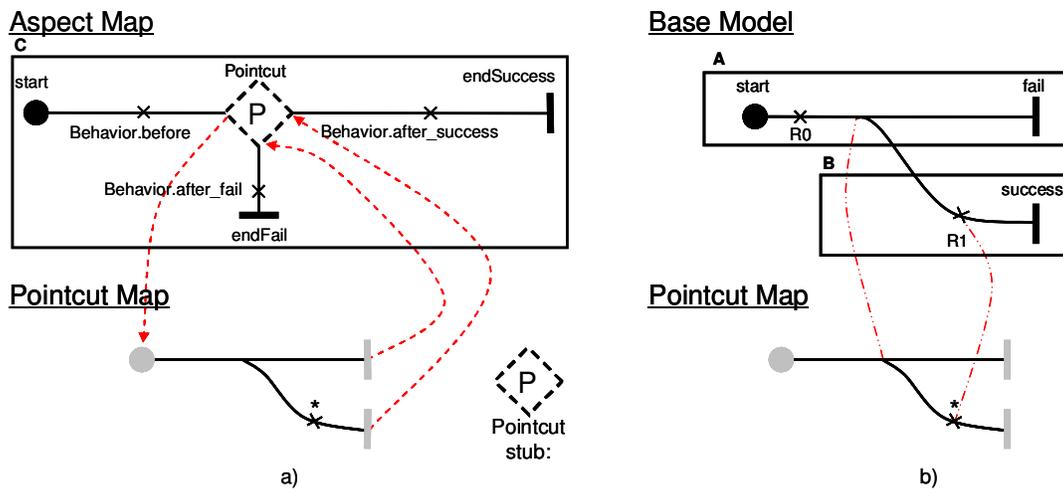


Figure 64 Pointcut Expressions of AoUCM Notation

Figure 65 contrasts the AoUCM style for the specification of pointcut expressions on the left with the AoGRL style on the right. On the right, the pointcut map and the aspect map are merged into one map, mimicking the pointcut graph approach in AoGRL. With the AoUCM style, it is easier to plug-in multiple pointcut expressions for the same concern or replace a pointcut expression if the concern is to be applied to another system. The AoGRL style is also a possible approach that, on the one hand, requires less management in terms of the plug-in bindings between the pointcut stub and the pointcut map, but on the other hand, causes duplication when several pointcut expressions are required for the

concern. Furthermore, the aspect map on the left is more generic than the aspect map on the right as it does not have to be altered when it is applied to a new system.

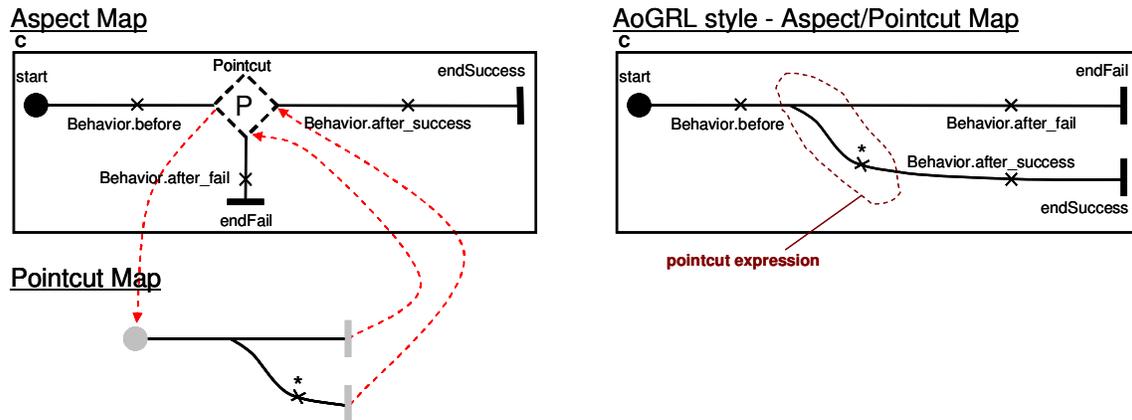


Figure 65 Comparing Pointcut Expressions for AoUCM and AoGRL

Although not shown in Figure 64.a, a pointcut map may contain UCM components (see Figure 41 on Page 75 for an example). In fact, any behavioral or structural UCM modeling element may be used on a pointcut map, allowing a wide array of partial maps to be matched. For example, the pointcut expressions in Figure 66 match against a) all start points starting with s, b) all responsibilities, c) all waiting places named ready or starting with w, and d) all components starting with A. These examples are not complete pointcut maps – in fact, the first three examples are not even valid UCM models. They illustrate only expressions which may be used on a pointcut map. The remaining examples in Figure 66 and all examples in Figure 67, on the other hand, show complete pointcut maps. See Appendix C: BNF for Name Expressions for a more formal definition of names in pointcut expressions.

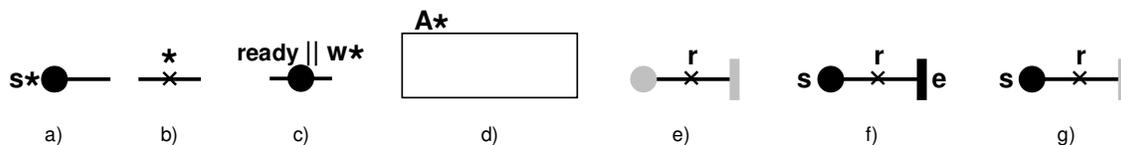


Figure 66 Examples of Visual Pointcut Expressions

Figure 66.e matches against all maps with a responsibility r, while Figure 66.f matches against all maps with a start point s, followed by responsibility r, and followed by an end point e. Figure 66.g, on the other hand, matches against all maps with a start point s followed by responsibility r. Note that it is possible to use unnamed start and end points as

markers for the start and end of the pointcut expression because the usage of unnamed start and end points is strongly discouraged in the UCM notation for increased clarity of the scenario itself, scenario definitions, and plug-in bindings. If the pointcut expression does not specify a name for a model element except responsibilities, components, start points, and end points, the name is deemed to be the wildcard *.

A pointcut expression is invalid, if it contains only a grey start point and a grey end point but no other type of path node and the start point and end point are both bound to the same component or not bound at all. Such a pointcut expression is invalid because it does not specify the required behavioural context for the match. Furthermore, a pointcut expression is also invalid, if at least one in-path or out-path of the pointcut stub is not bound to the pointcut expression. If an in-path or an out-path is not bound, composition of the concern cannot proceed. Furthermore, a pointcut stub without a plug-in map is not invalid, but equivalent to the concern not being applied as composition cannot proceed.

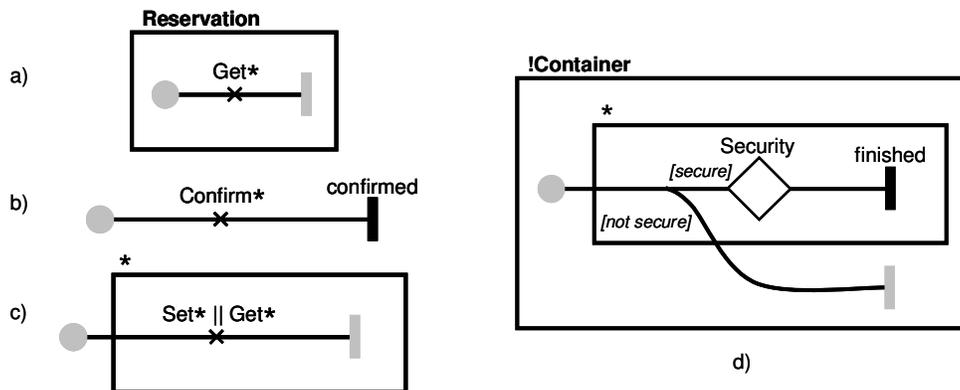


Figure 67 Examples of Pointcut Maps

Finally, the location of start and end points is also important for the meaning of the pointcut map. Figure 67.a matches all maps with a responsibility starting with Get and bound to the component Reservation. Figure 67.b matches all maps with a responsibility starting with Confirm. The responsibility must be immediately followed by an end point called confirmed. The responsibility and the end point may or may not be bound to a component. Figure 67.c matches all maps with a responsibility starting with Set or Get and bound to any component as the first path node of the component (because the start point is outside the component). Finally, Figure 67.d matches all maps with an OR-fork with conditions [secure] and [not secure] and bound as the first path node to any component

inside a component other than Container (i.e., !Container). The OR-fork must be immediately followed on the [secure] branch by a static stub called Security and an end point called finished, and followed by nothing on the [not secure] branch before exiting the component (because the end point is outside the inner component).

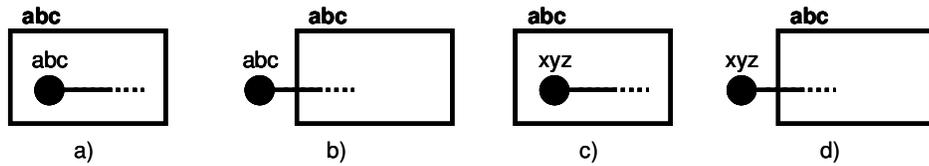
In other words, if a path on the pointcut map crosses the boundary of a component because of the location of a start or end point, then the path in the matching UCM will also have to cross the boundary of the matching component. Table 5 and Figure 68 provide a summary of all cases. The base maps and pointcut maps assume that after the start points all maps are identical. Therefore, whether a base map matches a pointcut map is solely dependent on the locations and the names of the start points. Note that the same matching rules apply to end points.

Table 5 Matching Rules for Start Points on Pointcut Maps and Base Maps

Do the base map and the pointcut map match?	Base map's start point ¹⁾ is <i>named</i> and <i>inside</i> component (Figure 68.a)	Base map's start point is <i>named</i> and <i>outside</i> component (Figure 68.b)	Base map's start point is <i>named</i> differently and <i>inside</i> component (Figure 68.c)	Base map's start point is <i>named</i> differently and <i>outside</i> component (Figure 68.d)
Pointcut map's start point ¹⁾ is <i>named</i> and <i>inside</i> component (Figure 68.e)	yes	no ²⁾	no ³⁾	no ³⁾
Pointcut map's start point is <i>named</i> and <i>outside</i> component (Figure 68.f)	no ⁴⁾	yes	no ³⁾	no ³⁾
Pointcut map's start point is <i>unnamed</i> and <i>inside</i> component (Figure 68.g)	yes	yes ⁵⁾	yes	yes ⁵⁾
Pointcut map's start point is <i>unnamed</i> and <i>outside</i> component (Figure 68.h)	no ⁶⁾	yes	no ⁶⁾	yes

- 1) The same reasoning applies to end points for all sixteen cases.
- 2) The pointcut expression stipulates that there has to be a start point with name abc inside component abc. Therefore, there is no match if the start point in the base map is not inside component abc.
- 3) There is no match in this case because the names do not match.
- 4) The pointcut expression stipulates that there has to be a start point with name abc outside component abc. Therefore, there is no match if the start point in the base map is not outside component abc.
- 5) This pointcut expression does not require the path to cross component abc, but it also does not exclude it. Therefore, the base map is matched although the start point is outside of component abc.
- 6) This pointcut expression requires the path to cross the component because the start point is outside of component abc. Therefore, the start point in the base map cannot be inside component abc.

Location and Naming of Start Point on Base Map



Location and Naming of Start Point on Pointcut Map

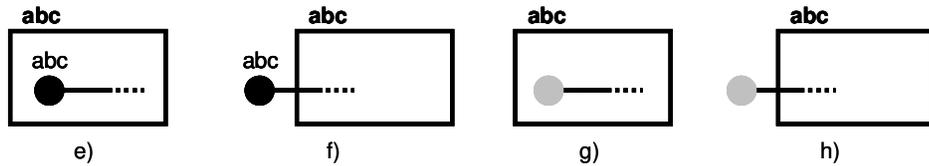


Figure 68 Location and Naming of Start Points on Pointcut Maps and Base Maps

In summary, the pointcut stub on the aspect map acts as a placeholder for the join points in the base model that are identified by the pointcut expression on the pointcut map. As a pointcut stub is a dynamic stub, it may contain many pointcut maps, thus making it possible to match very different patterns for the same aspect. In AOP, pointcuts can be rather lengthy and complex multi-line expressions. In aspect-oriented use case modeling, pointcuts reference extension points in a different use case. Both use text as the means to describe pointcuts. Considering the visual character of UCM and the pattern matching nature of aspect orientation, a visual representation for pointcuts is a much more natural choice. Note that the number of in-paths and out-paths for a pointcut stub is flexible. This flexibility makes it possible to take into account path elements with more than one insertion point before or after the path element (e.g., the OR-fork in Figure 64.a; each incoming or outgoing branch of the OR-fork can be matched individually).

Extensions to the URN Metamodel – The extensions to the URN metamodel required to support the pointcut stub concept are minimal (Figure 69), as only the pointcut attribute needs to be added to Stub.

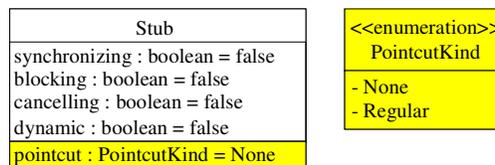


Figure 69 URN Metamodel Extensions: Pointcut Stub

6.3.4 Composition Rules

In AoUCM, the causal relationship of the pointcut stub and the aspectual properties visually defines the *composition rule* for the aspect, indicating how the aspect is inserted in the base model. In a continuation of Figure 64, Figure 70 again shows Behavior.before before the pointcut stub. Therefore, this responsibility must be inserted before the join points specified by the pointcut expression. Similarly, Behavior.after_success and Behavior.after_fail are inserted after the matched join points because they are shown after the pointcut stub. Note that the pointcut expression has two end points and therefore identifies two join points in the base model after which aspectual properties may be inserted. The plug-in binding of the out-paths of the pointcut stub to the end points on the pointcut map clearly defines the join point for the insertion of Behavior.after_success and the join point for the insertion of Behavior.after_fail. Since the end point with the responsibility on the branch of the OR-fork on the pointcut map is connected by the plug-in binding with the success out-path of the pointcut stub on the aspect map, the success case occurs on the branch of the matched OR-fork after an also matched responsibility. The fail case, on the other hand, occurs directly after the matched OR-fork on the other branch of the OR-fork.

Figure 70 indicates the matched pattern specified by the pointcut map in the base model. In addition, the corresponding parts of the base model, aspect model, pointcut map, and the plug-in bindings between the pointcut stub and the pointcut map are highlighted for each insertion of aspectual behavior into the base model. Elements related to the insertion before the matched pattern are circled in green, for the insertion in the success case in red, and for the insertion in the fail case in blue. Hence, the start point on the pointcut map is circled in green because it represents the location in the base model before the pattern which is also circled in green. Behavior.before is also circled in green as it describes what needs to be inserted. Finally, the plug-in binding that formally connects the location before the match with Behavior.before is also shown in green. Similarly, the end point on the pointcut map is circled in red because it represents the location in the base model after the pattern where the responsibility is on the branch of the OR-fork. This location is also circled in red in the base model. Behavior.after_success is also circled in red as it describes what needs to be inserted in this case. Finally, the plug-in bind-

ing that formally connects the location after the match with Behavior.after_success is also shown in red. The same applies to the location in the base model after the pattern where the responsibility is not on the branch of the OR-fork. All relevant elements are highlighted in blue.

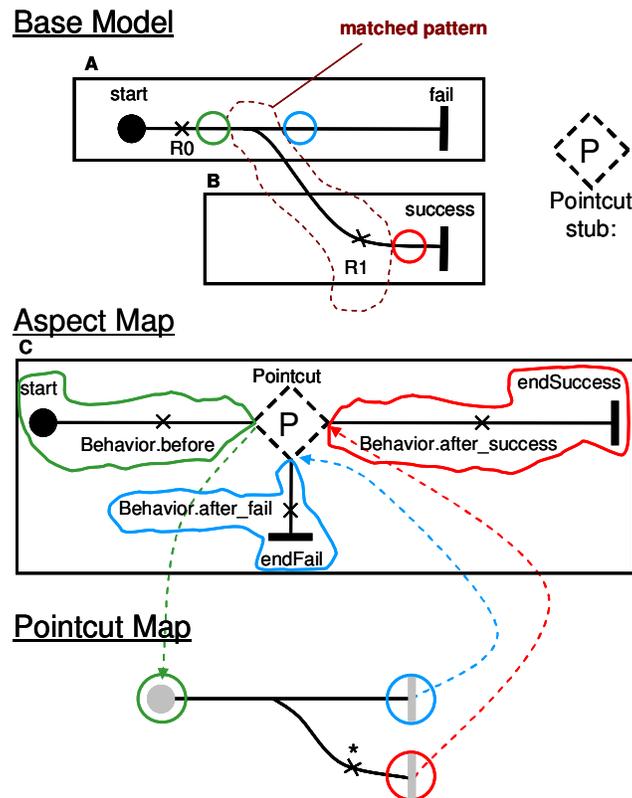


Figure 70 Composition Rules of AoUCM Notation

In addition to the simple sequential before and after composition rules shown in Figure 70.a, AoUCM can use any causal flow element and component structure in a composition rule, allowing for an exhaustive number of composition rules to be defined (for example but not limited to around, loop, concurrency, and interleaved composition rules). Due to the visual representation of the composition rule, a requirements engineer can understand at one glance the relationship of the aspect and the base model. For example, Figure 71.a shows very clearly that the aspect is being executed around the join points matched by the pointcut expression, allowing a situation to be modeled which occurs frequently in aspect-oriented modeling (i.e., the aspect overrides the base behavior). The aspect map in Figure 71.a shows that the join points are never executed in the composed system because the [false] branch is never taken. The [true] branch, however, is always taken and

therefore the responsibility Behavior.around is executed instead of the matched join points. Figure 71.b is an alternative and more concise representation of Figure 71.a. The *replacement pointcut stub* (❌) indicates that the aspect is replacing the matched join points with the responsibility Behavior.around. By convention, the replacement pointcut stub is always used right after the start point and has only one out-path. Figure 71.c depicts concurrent behavior, while Figure 71.d shows the aspectual behavior executing in a loop relative to the matched join points.

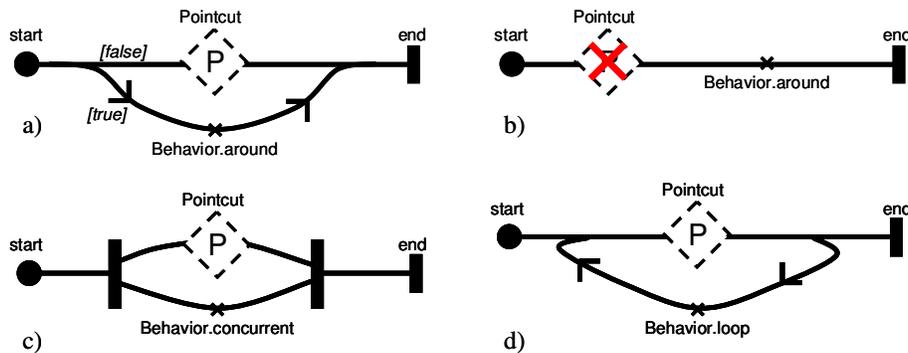


Figure 71 Examples of Composition Rules

As already stated in Section 6.1.4, the insertion points of the join points are indicated in the base model by small, filled diamonds called *aspect markers* (◆) which are added automatically to the AoUCM model by AoURN tools. Aspect markers can be thought of as *aspect stubs*, a special kind of static or dynamic stub that links to the portion of the aspect map that needs to be added at the insertion point. For example in a continuation of Figure 70, the aspect marker 1a in Figure 72.a links to Behavior.before, aspect marker 1b to Behavior.after_fail, and aspect marker 1c to Behavior.after_success. These links are also established automatically. In more detail, the plug-in binding of aspect marker 1a connects to the start start point and the in-path of the pointcut stub, aspect marker 1b to the fail out-path of the pointcut stub and the fail end point, and aspect marker 1c to the success out-path of the pointcut stub and the success end point. These plug-in bindings go beyond traditional plug-in bindings and require that extensions to the URN metamodel be supported. In contrast to static aspect stubs, dynamic aspect stubs are generally used when multiple aspects insert behavior at the same location in the model. Aspect markers that are inserted because of the same match of the pointcut expression (as is the case in Figure 72) refer to the same instance of the aspectual properties of the aspect. A second

set of aspect markers that is added because of a second match of the pointcut expression hence refers to a second instance of the aspectual properties of the aspect, unless the aspect map is defined as a singleton, in which case all aspect markers refer to the same instance of the aspectual properties of the aspect. The composed system is shown with a traditional UCM model in Figure 72.b with the help of regular stubs that are used to insert aspectual properties. For a more detailed description of how plug-in bindings of aspect markers are established automatically and for composition of AoUCM models in general, see Chapter 7.

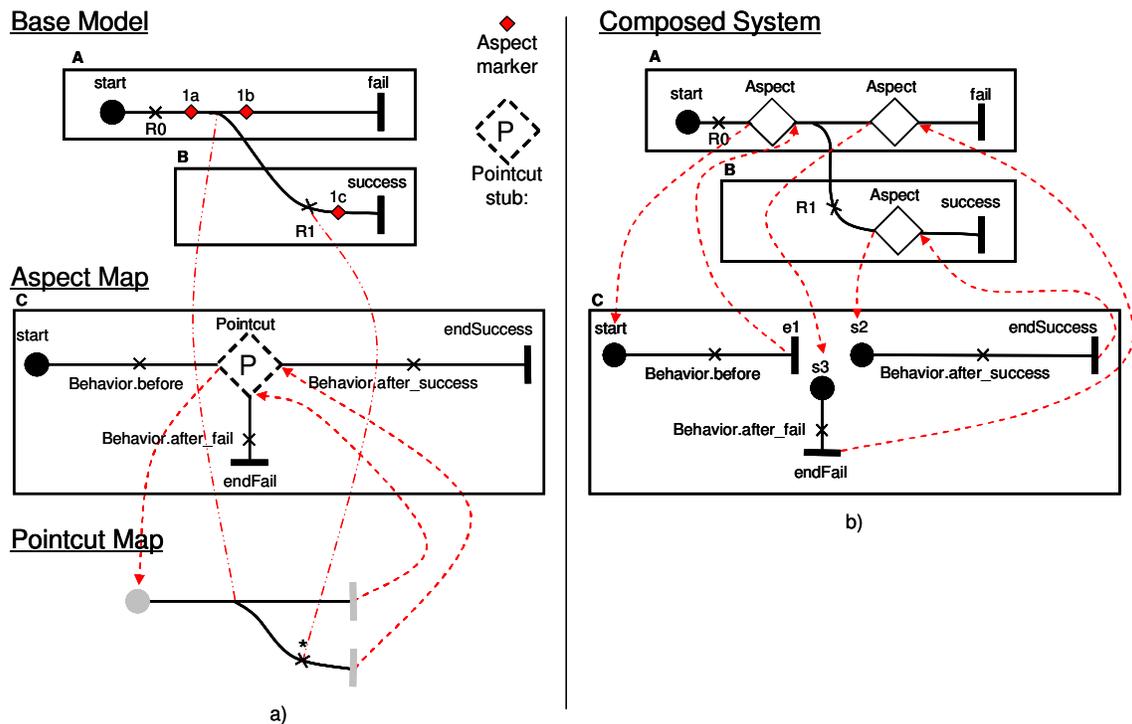


Figure 72 Aspect Markers of AoUCM Notation and Composed System

In contrast to AoGRL, pointcut expressions and composition rules are clearly separated in AoUCM as pointcut expressions are defined on pointcut maps and composition rules are specified on aspect maps. Therefore, pointcut expressions may be reused independently. A new UCM model may reuse a pointcut map by simply plugging it into a new pointcut stub. Similarly, the same aspect map may be reused in a different model with different pointcut expressions specific to the UCM model. For example, each aspect map in Figure 71 may have the same pointcut expression, because each pointcut stub may

contain the same pointcut map. The composition rules and aspectual behavior specified on the aspect maps, however, all differ.

Extensions to the URN Metamodel – The extensions to the URN metamodel required to support the aspect stub concept are fairly minor (Figure 73). For the aspect stub itself only the aspect attribute needs to be added to Stub. However, aspect stubs have a different plug-in binding mechanism which may involve the pointcut stub. This new plug-in mechanism has to be reflected in the URN metamodel, which currently allows only start and end points to take part in plug-in bindings. Currently, an InBinding connects an in-path of a stub on the parent map (stubEntry) to a StartPoint on the plug-in map. A new association now allows stubEntry also to be connected to the out-path of a pointcut stub (pointcutExit). The multiplicities of startPoint and pointcutExit are 0..1. An additional constraint ensures that an InBinding may use only one of them at a time. Similarly, an OutBinding currently connects an out-path of a stub on the parent map (stubExit) to an EndPoint on the plug-in map. A new association now allows stubExit also to be connected to the in-path of a pointcut stub (pointcutEntry). Again, the multiplicities of endPoint and pointcutEntry are 0..1. An additional constraint ensures that an OutBinding may use only one of them at a time.

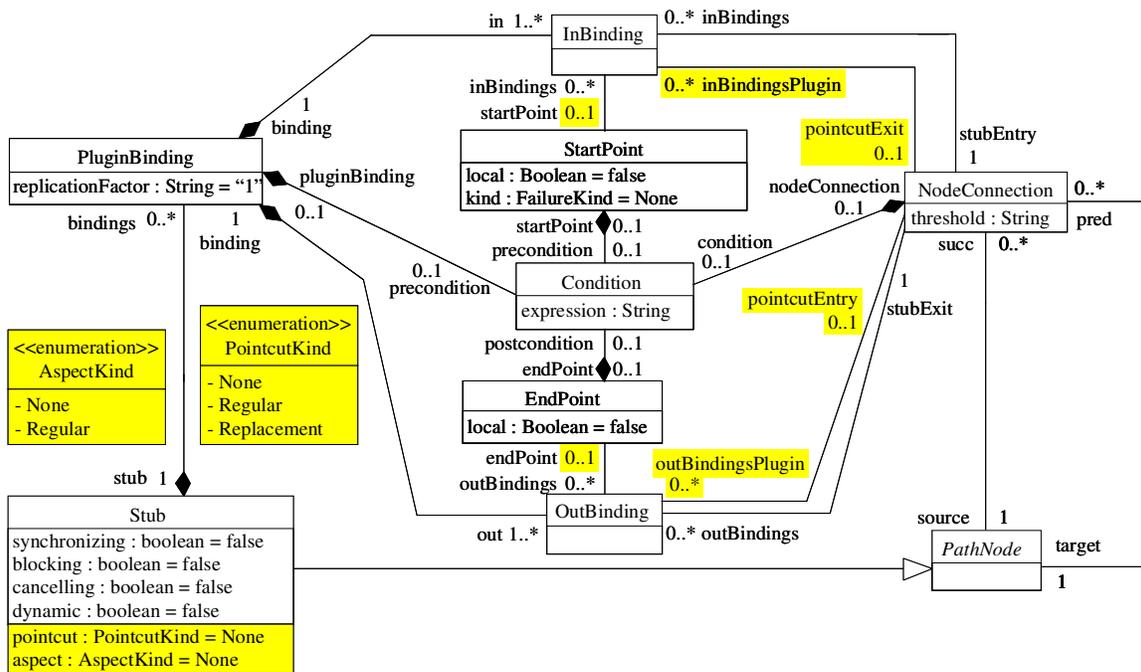


Figure 73 URN Metamodel Extensions: Pointcut Stub and Aspect Stub

Since regular pointcut stubs or replacement pointcut stubs may be used, the type of the pointcut attribute of Stub reflects these choices. The type is an enumeration called PointcutKind and may have the values None (not a pointcut stub), Regular (a regular pointcut stub), or Replacement (a replacement pointcut stub).

6.4. Navigating AoURN Models with AoViews

While the composition of AoURN models into standard URN models is often necessary to gain an overall understanding of the system, a more lightweight approach of navigating the aspect-oriented model itself allows the requirements engineer to reason about the model in an aspect-oriented way. Any AoURN tool must retain the aspect markers (i.e., the insertion points) and the mappings from pointcut expressions to the matched join points to facilitate navigation and reasoning about the AoURN model in an aspect-oriented way as shown in Figure 74 and Figure 75. The *AoViews* make use of the aspect diagrams to visualize to the requirements engineer the impact of applying an aspect at the insertion point identified by an aspect marker. AoViews can be generated without having to resolve potentially complex layout issues, because AoViews require only a simple highlighting capability of parts of existing aspect diagrams.

Given the base model, aspect graph, and pointcut graph in Figure 74, double-clicking on the aspect marker of Stakeholder Softgoal A1 or Stakeholder Goal C1 takes the requirements engineer to the AoViews of the pointcut graph on the right hand side of Figure 74. For Stakeholder Softgoal A1, the top AoView highlights Pointcut Specific Softgoal and its contribution as well as the correlation with the pointcut deletion marker, as these are transforming Stakeholder Softgoal A1. For Stakeholder Goal C1, Task of Aspect and its decomposition as well as the correlation with the pointcut deletion marker are highlighted in the bottom AoView.

Furthermore, the AoView replaces any parameterized element in the pointcut expression of the pointcut graph with the actual match from the base model. Therefore, the actor * in the pointcut graph is shown as Stakeholder, the softgoal * is shown as Stakeholder Goal A1 and the softgoal * Softgoal B is shown as Stakeholder Softgoal B. Pointcut deletion markers are still shown in the AoView to indicate the removal of model elements.

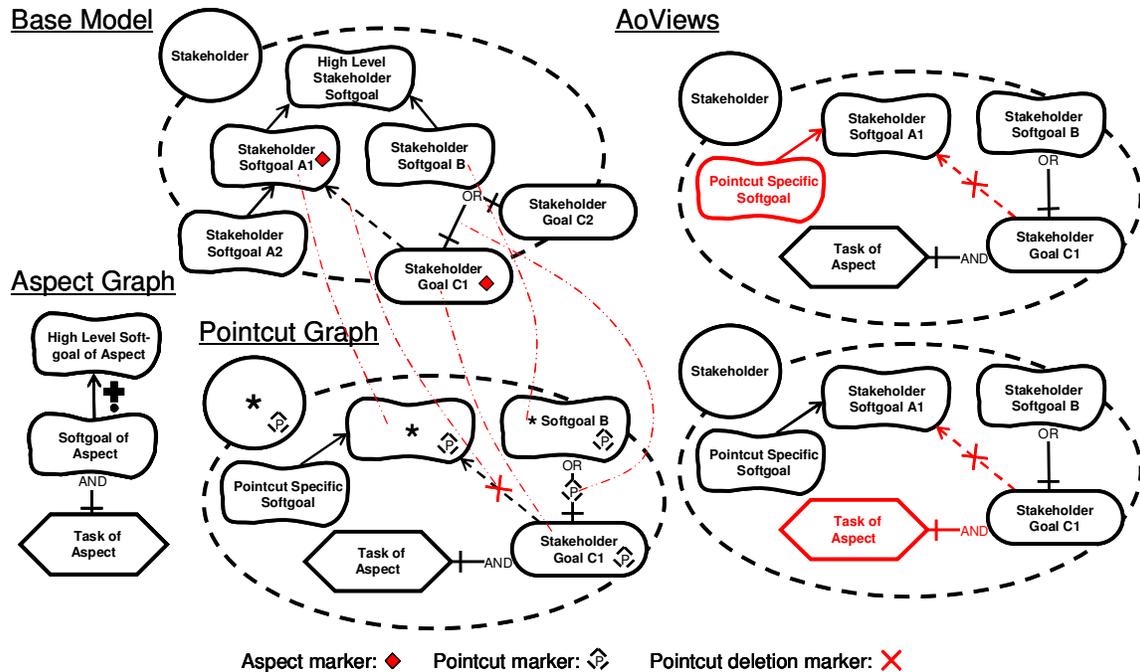


Figure 74 Navigating AoGRL Models with Aspect Markers and AoViews

If more than one aspect is applied to the location of the aspect marker, the requirements engineer first chooses an aspect from a list of all aspects applied to the selected location. Multiple aspects could even be selected and visualized with individual AoViews in different colors. This visualization applies to AoGRL as well as AoUCM.

The navigation of aspect-oriented UCM models is similar to the navigation of aspect-oriented GRL models. For example, double clicking on aspect marker 1a, 1b, or 1c in the base model, presents the requirements engineer with the AoViews of the aspect map on the right hand side of Figure 75. The aspect map's path segment that is connected to the chosen aspect marker through plug-in bindings established by the matching process is highlighted as shown in the navigation view. The AoView for aspect marker 1a is shown on the top (path segment with Behavior.before is highlighted), for aspect marker 1b in the middle (path segment with Behavior.after_fail is highlighted), and for aspect marker 1c at the bottom (path segment with Behavior.after_success is highlighted). Note that the arrows from the aspect markers to elements on the aspect map on the left hand side of Figure 75 visualize the plug-in bindings.

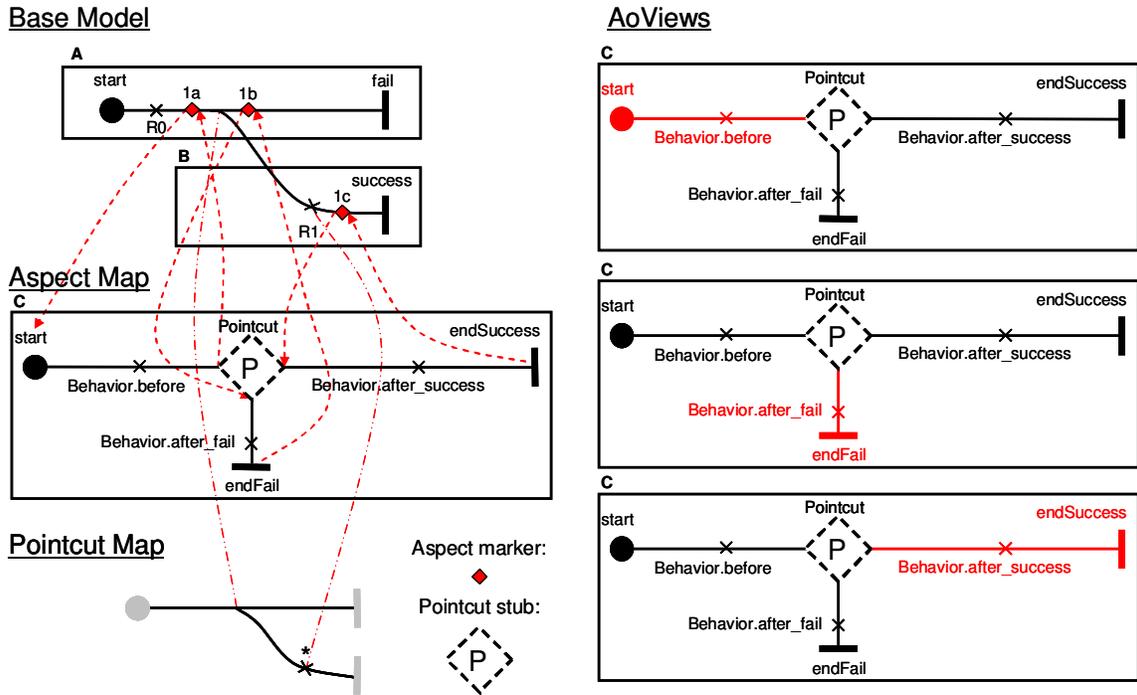


Figure 75 Navigating AoUCM Models with Aspect Markers and AoViews

6.5. Advanced Features of AoURN

6.5.1 Overview

While the features of AoURN introduced in Sections 6.1 to 6.4 enable the modeling of a wide variety of aspect-oriented goal and scenario models, five situations require more advanced features. The first situation applies to AoGRL and AoUCM models while the remaining four apply only to AoUCM models.

1. AoGRL and AoUCM pointcut expressions that specify the exact type of GRL intentional element, GRL link, or UCM path node are quite restrictive in that it is not possible to match patterns where some variations are allowed. For example, one may want to match against two responsibilities but does not care if a third responsibility exists between the first two. The *anything* and *anytype pointcut elements* address this issue as described in Section 6.5.2.
2. Often, AoUCM aspects need to add behavior to existing components. Given the ability of pointcut expression to use wildcards, one pointcut expression may match several components making it impossible to explicitly identify the

matched components on the aspect map. Hence, a better way to reuse matched components (and other elements) from the pointcut map should allow for the specification of much more generic aspect maps. For example, an aspect may want to add actions for a more elaborate security mechanism to scenarios of a particular set of users while not requiring them for other users. *Pointcut variables* address this issue as described in Section 6.5.3.

3. AoUCM aspects may insert new behavior in the UCM model but may also replace existing behavior. Aspect markers help a requirements engineer to understand the impact of an aspect on the scenario at hand. However, current aspect markers do not differentiate if the aspect causes straightforward insertions or more invasive replacements, even though there is a huge difference between these two changes. Additional concrete syntax for aspect markers called *tunnel aspect markers* and *conditional aspect markers* addresses this issue by identifying replacements more clearly as described in Section 6.5.4.
4. When a concern is applied to an AoUCM model, its pointcut expression is matched, and aspect markers are added automatically to the AoUCM model based on the match. Furthermore, plug-in bindings between the aspect marker and the aspect map must be established automatically. Sometimes, however, this may be ambiguous. For example, a redundancy aspect may have two end points, one for the case where it is possible to switch to the secondary system and one for the fail case. Both end points are potential candidates for the plug-in binding with the aspect marker. Hence, there needs to be a way to define which candidate to use. In AoUCM, *local* start and end points address this issue as described in Section 6.5.5.
5. Finally, two scenarios that are defined independently by two concerns may have to be interleaved when they are applied together. For example, a billing concern may have to add processing at the beginning of a telephone call as well as at the end of the call. Typically, this situation is resolved by creating two individual aspects with pointcut expressions that match the beginning and the end separately. This approach, however, leads to disjoint models which make it more difficult to grasp the overall behavior and intent. AoUCM sup-

ports *interleaved composition rules* to address this issue as described in Section 6.5.6.

6.5.2 Anything and Anytype Pointcut Elements

The anything and anytype pointcut elements allow a pattern to be matched where a portion of the pattern is open to variations in the match. AoUCM uses the *anything pointcut element* (denoted by `.....`) which allows any sequence of UCM model elements, including an empty sequence, to be matched. AoGRL uses the *anytype pointcut element* (`<<anytype>>`) which allows a single GRL intentional element or a single GRL link to be matched regardless of the type of intentional element or link, respectively. The reason why AoUCM and AoGRL are treated differently is due to the highly interconnected nature of GRL models. Allowing the anything pointcut element to be used in the context of AoGRL would lead to an explosion of matches because of the many and highly unstructured links between GRL intentional elements. Note that the ability to define the anything pointcut element is one of two extensions required for URN tools to specify AoUCM models (for the other see Section 6.3.3). The anytype pointcut element, on the other hand, is realized with the metadata name/value pair *aspect/anytype*. Any GRL intentional element or GRL link on a pointcut graph may be tagged with the *aspect/anytype* metadata.

As an example for the *anything* pointcut element, recall the Authentication concern from Section 5.4 repeated in Figure 76.a. Two aspect markers are added to the OVS system based on the pointcut expression defined as any responsibility on a path crossing from the Customer component into the OVS component. A new version of the OVS system, however, may require the parallel rewarding of the referrer in response to a request from the Customer as illustrated with the AND-fork and the rewardReferrer stub in Figure 76.b. Due to these modifications, the pointcut map in Figure 76.a does not match the OVS system anymore as it expects the responsibility to be the first path node in the OVS component. The pointcut map in Figure 76.b shows an improved pointcut expression that addresses this issue by making use of the anything pointcut element. This pointcut map matches the path crossing from the Customer component into the OVS component, the AND-fork, and the processOrder responsibility, because the anything pointcut element can be matched against the AND-fork. Note that the new pointcut map still

matches the second occurrence of the original pattern, i.e., the path crossing from the Customer component into the OVS component and the sendMovie responsibility because the anything pointcut element may be matched against nothing. The two aspect markers are hence still inserted as desired.

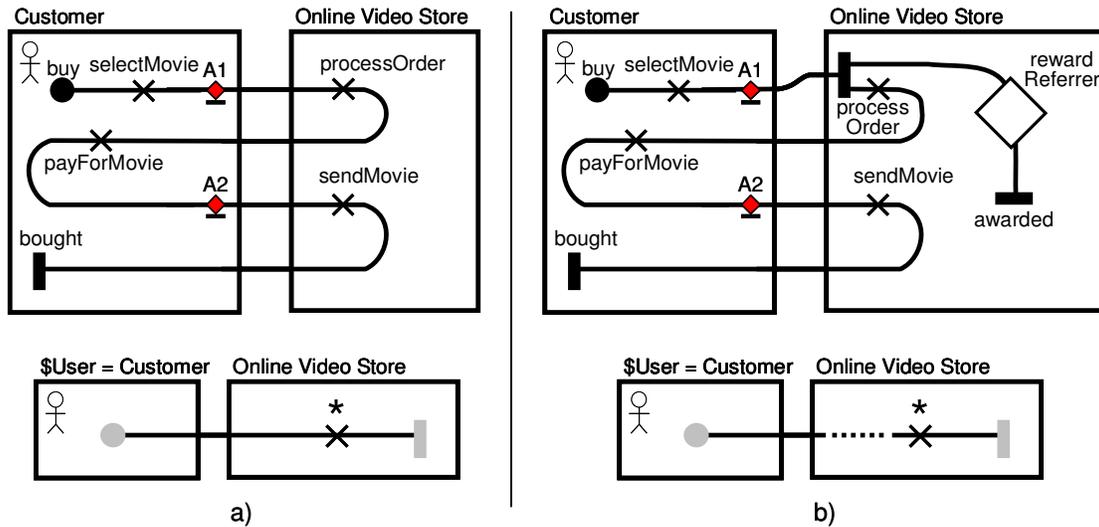


Figure 76 Allowing Variations with the Anything Pointcut Element

If several candidates for a match with an anything pointcut element exist, then only the shortest fully contained sequence of UCM model elements is matched. For example, the sequence consisting of “the path crossing into the OVS component – AND-fork – processOrder – path crossing back into Customer – payForMovie – path crossing into OVS component – sendMovie” is not matched, because the shorter sequence consisting only of “the path crossing into the OVS component – AND-fork – processOrder” is fully contained in the first sequence and also a valid match.

The location of the anything pointcut element relative to components is irrelevant for the matching process as the sequence matched by the anything pointcut element by default may contain elements bound to several components. This location irrelevance is necessary to be able to truly match any sequence of path elements. Figure 77 illustrates three pointcut expressions that all yield the same matching results.

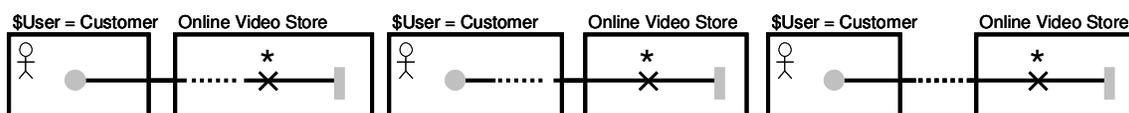


Figure 77 Location of Anything Pointcut Element on Pointcut Maps

A pointcut expression is invalid if it contains only a grey start point, the anything pointcut element, and a grey end point but no other type of path node, and the start point and end point are both bound to the same component or not bound at all. Such a pointcut expression would match against all path elements possibly within a component, leading to a large number of matches without any practical benefit. Furthermore, a pointcut expression with two consecutive anything pointcut elements is also invalid, as the same pointcut expression can be expressed with just one anything pointcut element.

As an example for the *anytype* pointcut element, recall the Authentication concern from Section 5.5 repeated in Figure 78.a. The rule encoded by the pointcut graph is that Security is an issue for an actor only if there exists a scenario for which security is an issue and the scenario has an impact on the actor. The pointcut graph in Figure 78.a, however, is more restrictive than this rule as it expects a contribution and a goal instead of allowing any kind of link and any type of intentional element as the rule suggests. The pointcut graph in Figure 78.b addresses this issue with two anytype pointcut elements. Now, the pointcut expression matches any type of link between the Order Movie task and an intentional element of any type in the actor. Note that there is no semantic difference whether a softgoal or goal or any other intentional element is tagged with `<<anytype>>`. The same applies to links.

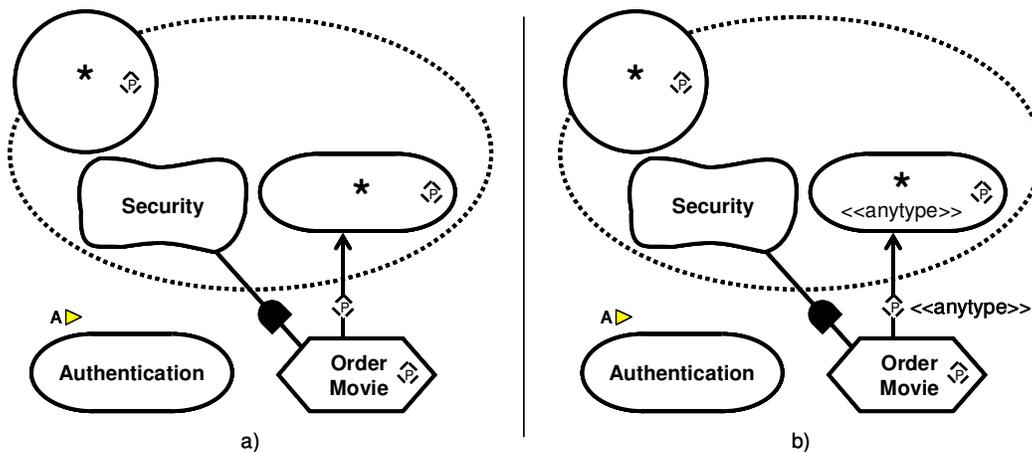


Figure 78 Allowing Variations with the Anytype Pointcut Element

A pointcut expression is invalid if it contains only links and intentional elements that are all tagged with the anytype pointcut element and the tagged model elements are all inside the same actor or not inside an actor at all. Such a pointcut expression would match

against any combination of elements possibly within an actor, leading to a large number of matches without any practical benefit.

Extensions to the URN Metamodel – The changes to the URN metamodel are rather small (Figure 79). A new path node *Anything* needs to be added. Constraints enforce that it is connected to exactly one previous and one next path node and that it can be used only on pointcut maps. Note that the binding of an anything pointcut element to a component is irrelevant.

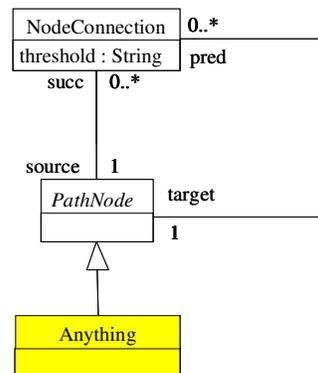


Figure 79 URN Metamodel Extensions: Anything Pointcut Element

6.5.3 Pointcut Variables

UCM models contain responsibility definitions and component definitions as the main building blocks for UCM scenario descriptions. Aspects can use these building blocks in the description of their aspectual properties, but sometimes it is not possible to explicitly define which component or responsibility should be used as this may depend on what is matched by the pointcut expression. A *pointcut variable* enables a matched responsibility or component from the UCM base model to be reused by the aspect in the description of its aspectual properties. Hence, there are two types of variables: responsibility variables and component variables.

First, a variable is defined for a responsibility or a component on the pointcut map of the aspect by assigning a variable name to the element. This assignment is done by adding a prefix starting with the \$ symbol to the element. For example in the pointcut map in Figure 80.b, the prefix \$action = is added to * on a responsibility, which means that the action variable represents all responsibilities that match *. Given the base model

in Figure 80.a, the action variable matches against R1 and R2. See Appendix C: BNF for Name Expressions for a more formal definition of variable expressions on pointcut maps.

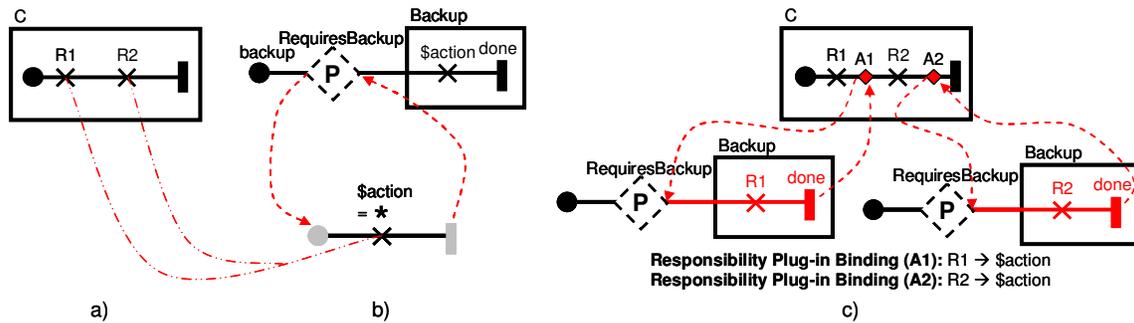


Figure 80 Reuse of UCM Model Elements with Variables

Second, the variable is used on the aspect map by adding an element with the variable name to the aspect map (i.e., \$action in Figure 80.b). A variable that is used by an aspect must be defined on all of the aspect's pointcut maps or the aspect is invalid. Furthermore, a variable must not be defined more than once on a pointcut map or the pointcut map is invalid. The type of the variable determines whether a responsibility or a component is used in the aspect map. The aspect map in Figure 80.b hence defines a simple Replication concern that performs all responsibilities on a Backup server.

When the aspect is finally applied to the base model as illustrated in Figure 80.c, the variable in the aspect map is replaced by the actual match for each match of the pointcut expression based on the mappings established by the matching process (long dash-dot-dotted lines between Figure 80.a and Figure 80.b). Given these mappings, the AoView of an aspect marker is then able to substitute the variable with the actual match for each aspect marker. To enable off-line visualization of the aspect, the substitution instructions required by the AoView must be encoded for the aspect marker. This encoding is realized by converting the mapping into responsibility plug-in bindings. For example, the responsibility plug-in binding for aspect marker A1 in Figure 80.c connects responsibility R1 in the base model with the responsibility \$action in the aspect map (i.e., the variable). AoURN's matching and composition algorithm automatically established these responsibility bindings. The analogous approach is used for component variables and component plug-in bindings.

As a further example for the definition of responsibility variables and component variables, recall the Communication concern in Figure 53 on Page 87. Examples of AoViews with variables are also discussed for Figure 55 and Figure 56.

Since pointcut expressions of an aspect may be matched against aspectual properties of another aspect, the following case needs to be considered: a pointcut expression of aspect A is matched against an aspect B that uses one or more variables. The actual values of a variable, however, are known only once aspect B is applied. It may therefore not be possible to determine at the time of matching whether a match exists or not. The match depends on the pointcut expression of aspect A and the definitions of the variables of aspect B.

In simple cases, it can be determined right away that a match exists. For example, the Logging concern matches against any responsibility of the Online Video Store component. Therefore, it can be matched against `$performRequest` of the Communication concern, because any value of `$performRequest` certainly matches `*`. Furthermore, it can also be matched against `$Replier`, because `$Replier` is defined as Online Video Store on the pointcut map of the Communication concern. Consequently, there is always a match (Figure 81.a).

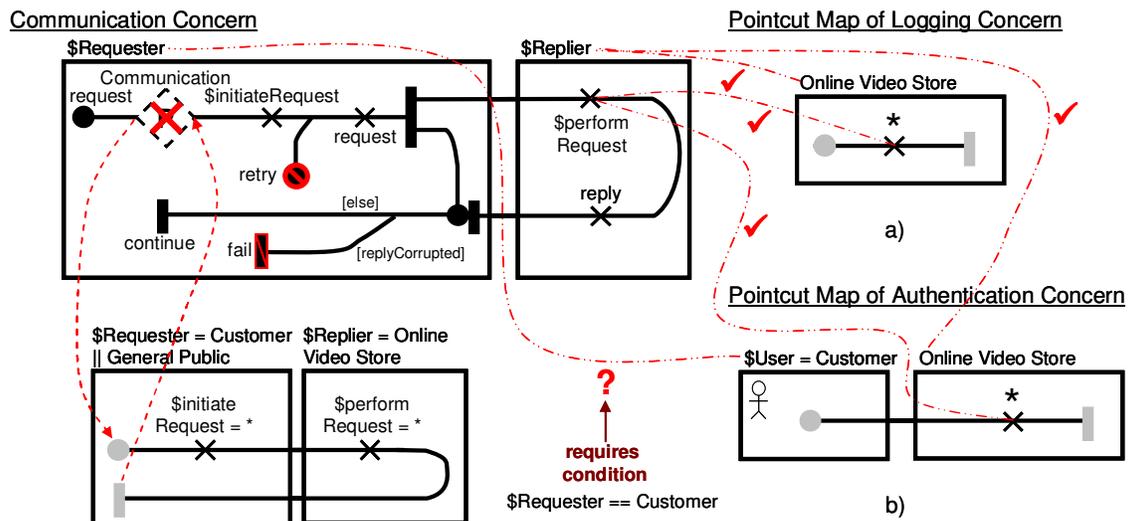


Figure 81 Matching of Variables

On the other hand, the Authentication concern cannot be matched against the Communication concern right away. The Authentication concern matches against any responsibility on a path crossing from the Customer component into the Online Video Store component.

As with the Logging concern, the responsibility can certainly be matched against $\$performRequest$ and Online Video Store can be matched against $\$Replier$. However, Customer cannot necessarily be matched against $\$Requester$, because $\$Requester$ is defined as $Customer \parallel General\ Public$. Therefore, this match requires a condition that encodes the requirements for the match, i.e., $\$Requester == Customer$ (Figure 81.b). This condition is added to the aspect markers associated with this match. If multiple variables need to be matched after composition, the condition will be a conjunction. The AoView of the Communication concern can then either show the aspect marker, if there is a match with the actual values of the variables (maps on middle left and bottom left in Figure 56), or not show the aspect marker, if there is no match with the actual values of the variables (maps on middle right and bottom right in Figure 56).

Coincidentally, this situation is similar to component and responsibility plug-in bindings which may be defined for a plug-in map that is matched by pointcut expressions. While it is always possible to determine statically whether a match exists, because the plug-in binding specifies unambiguously the replacement, the plug-in map may be used by several stubs and different plug-in bindings may therefore be specified. Consequently, a match may exist only in the context of a particular plug-in binding. Hence, aspect markers added to the UCM model because of the match may be required in one context but not the next. Hence, conditions similar to the conditions for variables explained above have to be added to the aspect markers.

Note that variables are required only for AoUCM and not for AoGRL because of the clear separation of pointcut maps and aspect maps. In AoGRL, the pointcut graph may contain new elements which can be simply added to matched elements, because they are both shown on the same pointcut graph. The assignment of new intentional elements to a parameterized actor can therefore be specified directly on the pointcut graph without a need for variables.

Extensions to the Traversal Mechanism – The traversal mechanism must take into account variables on aspect maps and substitute them with their actual matches with the help of component plug-in bindings and responsibility plug-in bindings, i.e., the traversal mechanism must use the AoViews when traversing the content of aspect markers.

Note that conditions for aspect markers do not require an extension because stubs already have conditions that are taken into account by the traversal mechanism.

6.5.4 Tunnel Aspect Markers and Conditional Aspect Markers

Four different types of aspect markers with different concrete syntax characterize the impact of an aspect on a scenario, making it possible for the requirements engineer to intuitively understand the interaction between the scenario and the aspect. The regular aspect marker (◆) indicates that the aspect inserts aspectual properties into the scenario and that the scenario continues after the aspect marker. The *tunnel entrance aspect marker* (◆) and *tunnel exit aspect marker* (◆) indicate that the aspect replaces a portion of the scenario with aspectual properties. The metaphor of a tunnel is used for the aspectual properties because they appear like a tunnel to the scenario. Instead of unfolding normally, the scenario is changed by unknown behavior hidden inside a tunnel that circumvents the original behavior of the scenario. The tunnel entrance aspect marker indicates the beginning of the tunnel to the scenario, while the tunnel exit aspect marker indicates the end. In other words, the original scenario does not continue past the tunnel entrance aspect marker, but continues only at the tunnel exit aspect marker. Finally, the *conditional aspect marker* (◆) indicates that the aspect may or may not continue past the conditional aspect marker.

As an example of tunnel entrance aspect markers and tunnel exit aspect markers, recall the Communication concern in Figure 55 on Page 88. The request/reply communication pattern replaces the original behavior in the base model with aspectual behavior. This replacement is clearly indicated on the aspect map with the replacement pointcut stub. The impact of the Communication concern is shown by the tunnel aspect markers. Each pair of tunnel aspect markers is labelled with the same name (e.g., C1). Conceptually, a pair of tunnel aspect markers corresponds to one regular aspect marker. The visualization, however, is much simplified if a pair of tunnel aspect markers is shown instead of removing the replaced base behavior and showing only a single aspect marker. Unfortunately, the pair of tunnel aspect markers violates a constraint of the traversal mechanism, because the traversal of a stub and its plug-in map expects to return to the same

stub once traversal of the plug-in map has finished. AoURN requires this constraint to be weakened for aspect markers.

Since aspects are applied many times due to their crosscutting nature, the same aspect map may be plugged into many tunnel aspect markers. It is therefore ambiguous to which aspect marker to return once traversal of the aspect map completes. Intuitively, the traversal returns to a tunnel aspect marker with the same name as illustrated in Figure 55. This intuition is correct since the tunnel aspect markers with the same name are inserted because of the same match of the pointcut expression and hence refer to the same instance of the aspectual properties. These tunnel aspect markers are said to belong to the same *aspect marker group*. Aspect marker groups must be clearly specified to support off-line visualization of the aspect with AoViews. An aspect marker group is realized with the metadata name/value pair *aspect/id group N* which is assigned to the tunnel aspect markers in a group. It states that the aspect marker belongs to group N of the aspect map with the given ID. For example, assuming that the ID of the aspect map for the Communication concern is 1234, then the aspect tunnel markers named C1 in Figure 55 have the metadata *aspect/1234 group 1*, while the aspect tunnel markers named C2 have the metadata *aspect/1234 group 2*, and so on. The metadata information is automatically added by AoURN's matching and composition algorithms.

The combined behavior of the Order Movie scenario and the request/reply communication pattern unfolds therefore as follows in Figure 55 on Page 88:

- buy start point,
- enter tunnel entrance aspect marker C1 [traversal takes note of aspect marker group 1 and substitution instructions `selectMovie` → `$initiateRequest`, `processOrder` → `$performRequest`],
- out-path of Communication pointcut stub,
- `selectMovie` (replacing `$initiateRequest`),
- `request` (the actual communication),
- AND-fork [on first branch: `processOrder` (replacing `$performRequest`), `reply` (the actual communication), on second branch: nothing],
- parallel branches synchronize at waiting place,
- [else] branch,

- continue end point (has four possible plug-in bindings to tunnel exit aspect markers C1, C2, C3, and C4),
- exit tunnel exit aspect marker C1 (because only C1 belongs to the same aspect marker group as the tunnel entrance aspect marker),
- the same again for tunnel aspect markers C2 (but with `payForMovie` and `sendMovie` replacing `$initiateRequest` and `$performRequest`, respectively, as well as aspect marker group 2),
- bought end point.

As an example for the conditional aspect marker, recall the Authentication concern in Figure 49 on Page 83. The conditional aspect marker is caused by a choice point in the aspect map that may lead the aspectual scenario to a local end point depending on the evaluation of the choice point's conditions. In the example in Figure 49, there are even two choice points that fulfill this criteria (the `blocked/else` OR-fork and the `> 3 times/< 4 times` OR-fork). If the local end point is reached in the aspectual scenario, then the scenario does not continue past the conditional aspect marker. If the pointcut stub is reached in the aspectual scenario, then the scenario will continue past the conditional aspect marker.

The loop composition in Figure 82.b is another example where the conditional aspect marker is required, as the visualization in Figure 82.c shows. The conditional aspect marker is required because the aspect map contains a choice point that may not lead the aspectual scenario to the looped end point that is connected to the exit of the conditional aspect marker L1. The scenario may therefore not continue past the conditional aspect marker. Only if the branch of the choice point with responsibility A3 is taken on the aspect map, the scenario does continue past the conditional aspect marker L1 when it reaches the end point.

Figure 82.c is also an example for a second set of aspect maps for which aspect marker groups are required. An aspect marker group ensures that plug-in bindings from the aspect map to aspect markers are effectively shared among all aspect markers in the group, because such plug-in bindings can be used to exit the aspect map regardless of which aspect marker is used to enter the aspect map. This is the case for the first set of aspect maps for which aspect marker groups are required (easily identified by their re-

placement pointcut maps and tunnel aspect markers), and it is also the case for loop composition as demonstrated in Figure 82.c. The traversal mechanism may enter the aspect map through the conditional aspect marker L1 and exit the aspect map through the regular aspect marker L1, if the branch with responsibility A2 is taken. Hence, the plug-in binding of the regular aspect marker L1 is shared with the conditional aspect marker L1, which requires an aspect marker group. Figure 82.c shows the metadata defining the aspect marker group for the two aspect markers, assuming that the ID of the aspect map is 15.

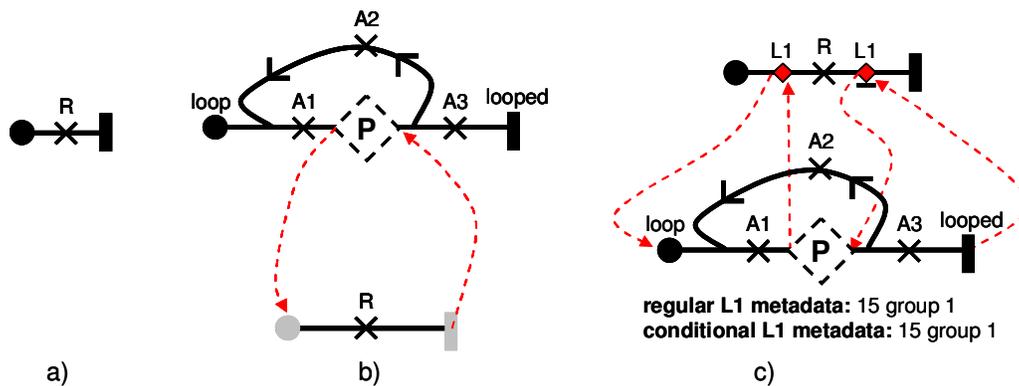


Figure 82 Loop Composition

This second set of aspect maps for which aspect marker groups are required is identified by the structure of the aspect map and its aspect markers associated to one match. If it is possible to reach an element that does not have an out-binding to the same aspect marker used to enter the plug-in map, but the element has an out-binding to a different aspect marker, then an aspect marker group is necessary. Typically, this is the case for aspect maps with choice points or parallel branching that make use of conditional, loop, or concurrency composition rules. For a more detailed description of the matching and composition mechanism of AoUCM models, see Chapter 7.

Extensions to the Traversal Mechanism – The traversal mechanism must weaken one of its constraints by taking into account aspect marker groups for traversal between aspect markers and their plug-in maps. Traversal may now enter a plug-in map through one aspect marker and exit the plug-in map through another aspect marker as long as the aspect markers belong to the same aspect marker group, i.e., the traversal mechanism must use the AoViews when traversing the content of aspect markers.

Extensions to the URN Metamodel – The extensions to the URN metamodel required to support the aspect stub concept are fairly minor (Figure 83) as only the aspect attribute needs to be added to Stub. However, since various kinds of aspect stubs may be used, the type of the aspect attribute of Stub reflects these choices. The type is an enumeration called AspectKind and may have the values None (not an aspect stub), Regular (a regular aspect stub), Entrance (a tunnel entrance aspect stub), Exit (a tunnel exit aspect stub), or Conditional (a conditional aspect stub).

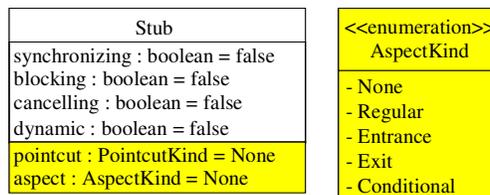


Figure 83 URN Metamodel Extensions: Aspect Stub

6.5.5 Local Start and End Points

Figure 63 on Page 102 illustrates the insertion points of a UCM model. It is no coincidence that small diamonds are used to indicate the insertion points as well as aspect markers, since aspect markers identify insertion points. Hence, the insertion points in Figure 63 also represent potential aspect markers. A quick glance at that figure shows that all aspect markers have only one in-path and only one out-path. Actually, an aspect cannot introduce new paths because this would lead to a violation of the separation of concerns the aspect wants to achieve. If, for example, an aspect marker is allowed to introduce a second out-path, the second out-path could be used by the scenario to which the aspect is applied, only if the scenario knew about the second out-path. However, separation of concerns dictates that the scenario does not know about the aspect in the first place and hence cannot know about an out-path introduced by the aspect. Consequently, aspect markers have only one in-path and one out-path. This fact, however, restricts aspect maps, because plug-in bindings have to be established automatically between an aspect marker and an aspect map. If the aspect map contains more than one possible candidate for the plug-in bindings with the one in-path and the one out-path of the aspect marker, those plug-in bindings cannot be established automatically.

Because only one in-path and one out-path must exist for an aspect marker to ensure proper encapsulation of concerns, there are only two options for the plug-in binding of an aspect marker for each of its plug-in maps. By definition, either the in-path of the aspect marker is bound to a start point and the out-path of the aspect marker is then bound to the in-path of the pointcut stub, or the in-path of the aspect marker is bound to the out-path of the pointcut stub and the out-path of the aspect marker is then bound to an end point (Figure 84.a). Since the plug-in bindings of an aspect marker are established automatically, there can be only one start point and only one end point from which to choose for the plug-in bindings (Figure 84.b). Therefore, there can be only one start point from where the in-path of the pointcut stub can be reached. If multiple start points exist from where the in-path can be reached, then the plug-in binding cannot be established automatically, because it is no longer clear which start point to bind. Similarly, there can be only one end point that can be reached from the out-path of the pointcut stub. If multiple end points exist that can be reached from the out-path, then the plug-in binding again cannot be established automatically, because it is not clear which end point to bind. *Local start points* and *local end points* help identify the start point and end point that should be used for the plug-in binding.

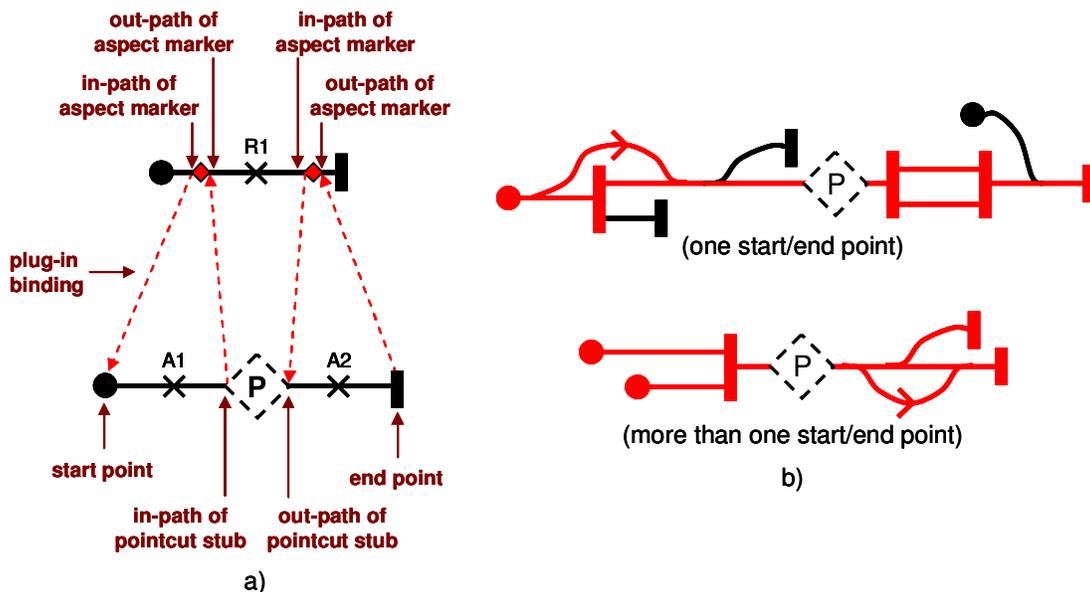


Figure 84 Plug-in Bindings for Aspect Markers and Candidates on Aspect Maps

As an example of local start and end points, recall the Communication concern in Figure 53 on Page 87. Further examples are shown in Figure 85 to indicate when local start and end points must be used. The first set of examples in Figure 85.a shows aspect maps for which local start or end points are not required. The in-path of the pointcut stub can always be reached trivially from only one start point, since there exists only one start point on the map. Similarly, only one end point can be reached trivially from the out-path of the pointcut stub, because only one end point exists on the map.

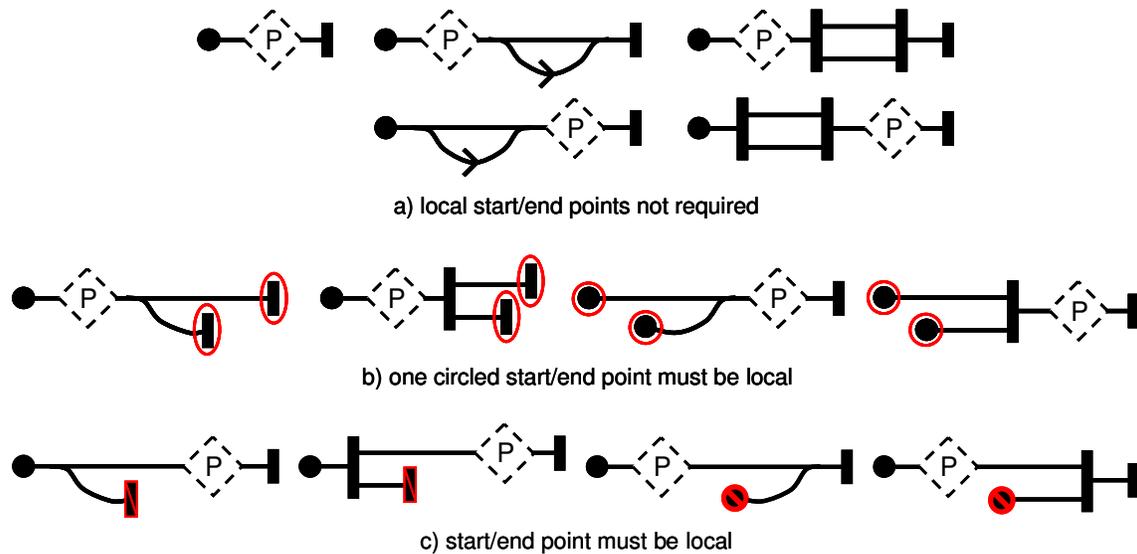


Figure 85 Local Start and End Points on Aspect Maps – Part I

The second set of examples in Figure 85.b indicates with red circles where multiple options for the plug-in bindings exist. For example, one can clearly see that a traversal of the path in a forward direction from the out-path of the pointcut stub in the left-most example in Figure 85.b reaches both end points. Therefore, one of the circled start or end points in a map must be designated as a local start or end point, respectively, stating that it is not to be used for the plug-in binding.

The third set of examples in Figure 85.c has only one start point from where the in-path of the pointcut stub can be reached and one end point that can be reached from the out-path of the pointcut stub. However, additional start and end points exist on the map that do not reside on the path from the start point to the in-path or on the path from the out-path to the end point. These start and end points must be designated as local, because plug-in bindings will never be specified for them. They must be designated as local

to clearly indicate to the requirements engineer that the base behavior will not be connected with such start and end points.

Any aspect map in Figure 85 where the pointcut stub is placed right after the start point could have a replacement pointcut stub instead of the regular pointcut stub. The reasoning about local start and end points remains exactly the same as for regular pointcut stubs because, conceptually, a pair of tunnel aspect markers corresponds to one regular aspect marker. The only difference is that, while a regular aspect marker has one plug-in binding to the aspect map and another one from the aspect map, a pair of tunnel aspect markers splits up these plug-in bindings among the two tunnel aspect markers.

While the examples in Figure 85 show pointcut stubs with only one in-path and only one out-path, the same kind of reasoning for local start and end points applies to pointcut stubs with multiple in-paths or out-paths (Figure 75 on Page 116). As a general rule, one can expect one non-local start point per in-path of the pointcut stub and one non-local end point per out-path of the pointcut stub.

The examples in Figure 85 also show only aspect maps where the pointcut stub partitions the aspect map into disjoint path segments. Nevertheless, the same reasoning still applies to aspect maps for which this restriction does not hold, as the examples in Figure 86 indicate.

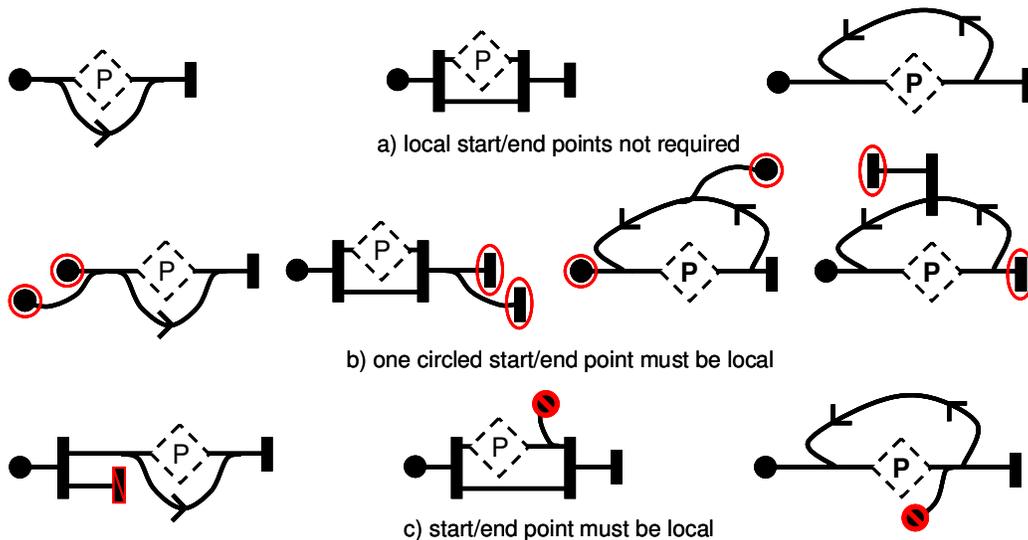


Figure 86 Local Start and End Points on Aspect Maps – Part II

Figure 87 show pointcut stubs with more than one in-path or out-path on an aspect map that is not partitioned into disjoint path segments by its pointcut stub. While the same reasoning generally still applies, some of these examples belong to a category of special cases (Figure 87.c). In the top left example in Figure 87.c, the end point reached from the OUT1 out-path is circled in green. Both end points, however, can be reached from the OUT2 out-path. Nevertheless, there is no need for a local end point in this case, because an aspect marker group will be created for this case as per the definition at the end of Section 6.5.4. The aspect marker group ensures that the plug-in bindings of end points are effectively shared among all aspect markers in the group, because such plug-in bindings can be used to exit the aspect map regardless of which aspect marker is used to enter the aspect map. The OUT2 out-path reaches the green and the red end point. However, since the green end point is already dealt with by the OUT1 out-path, there is no need to establish the plug-in binding for the end point again as it is already available through the aspect marker group. Therefore, the only candidate left for the OUT2 out-path is the red end point. The same reasoning applies to the remaining examples in Figure 87.c. While the top right example does not require any local start points, the bottom examples require one of the two red start/end points to be designated as local.

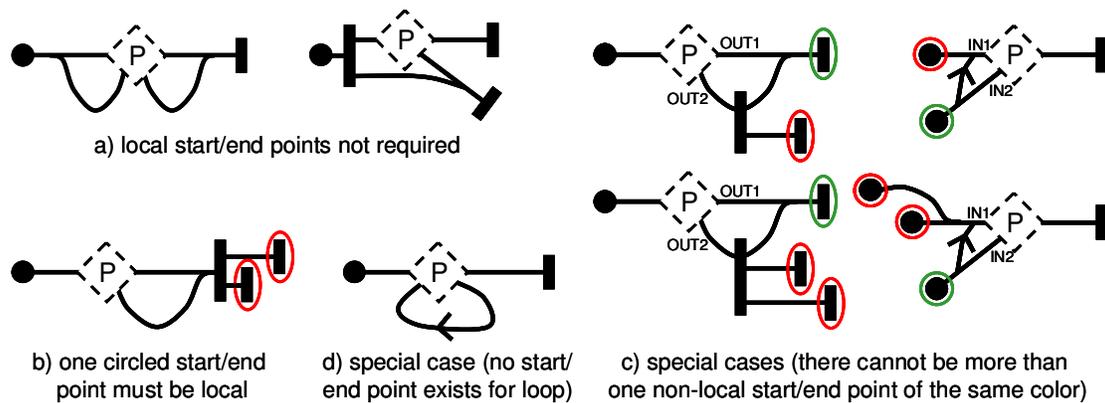


Figure 87 Local Start and End Points on Aspect Maps – Part III

Finally, the example in Figure 87.d is the last special case. It involves in-paths or out-paths of the pointcut stub that loop back to the pointcut stub. In this case, no corresponding start or end point exists for the in-path or out-path of the pointcut stub. Since plug-in bindings for the in-path and out-path of the pointcut stub are established by default, there

is no need to create further plug-in bindings. Therefore, tunnel aspect markers with either only one in-binding or only one out-binding are used to indicate this situation.

6.5.6 Interleaved Composition Rules

Interleaved composition occurs when two independent scenarios have to be merged while maintaining a certain partial ordering, e.g., part 1 of scenario 1 executes first, then part 1 of scenario 2, then part 2 of scenario 1, and so on. Since the scenarios are independent, they should be modeled by two concerns. AoURN enables *interleaved composition rules* for concerns by allowing multiple pointcut stubs on one aspect map. Each of these pointcut stubs has the same pointcut map as its plug-in map.

To illustrate an example of interleaved composition rules, recall the Order Movie scenario from Figure 40 and repeated on the left side of Figure 88. The right side of Figure 88 shows another scenario of the OVS system. It involves the earning and redeeming of movie points. First, the Customer is given the choice to become a member of the movie club which is a prerequisite for earning and redeeming movie points. A member may then use her movie points for a purchase, if enough points are accumulated. Finally, a member who does not redeem points for a purchase may earn movie points on a purchase which are credited by the OVS to the member's account. The UCM model of the Movie Points scenario clearly conveys the causal relationships of the three responsibilities in the scenario.

While the Order Movie and Movie Points scenarios can be understood in isolation, they need to be eventually merged together. The desired combined scenario is shown in Figure 89. The member form needs to be filled out before processing the order, movie points may be redeemed instead of paying for a movie, and movie points may be credited only once the movie is sent.

It is not possible to model the Order Movie concern and the Movie Points concern on separate maps with traditional UCM without breaking up one of the concerns into several disjoint paths and losing its contextual information. Many aspect-oriented techniques even break up the Movie Points concern into three separate sub-concerns and apply each of them individually. This approach allows the concerns to remain separate, but contextual information is again lost.

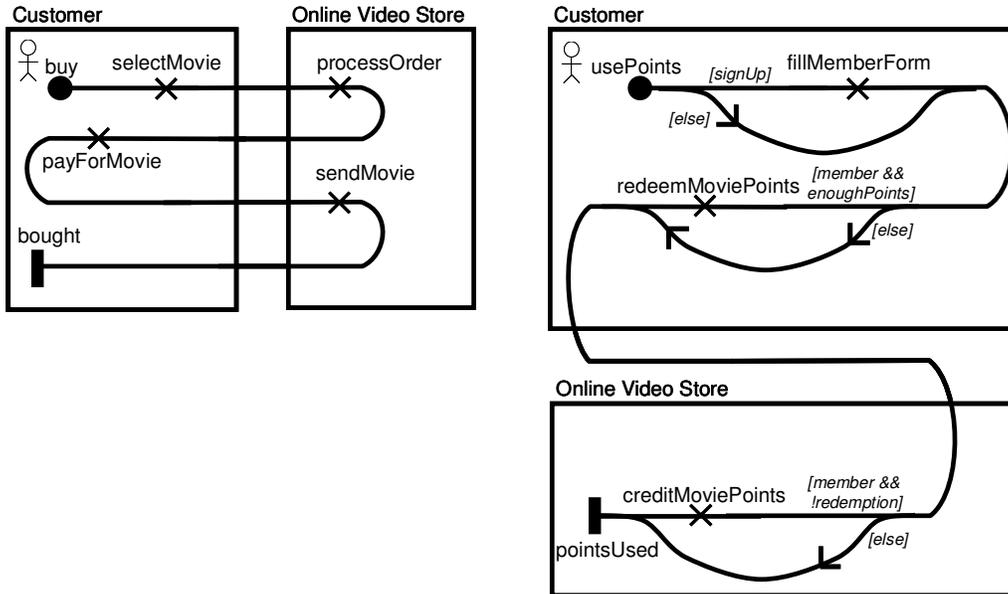


Figure 88 Movie Points Scenario of the Online Video Store System

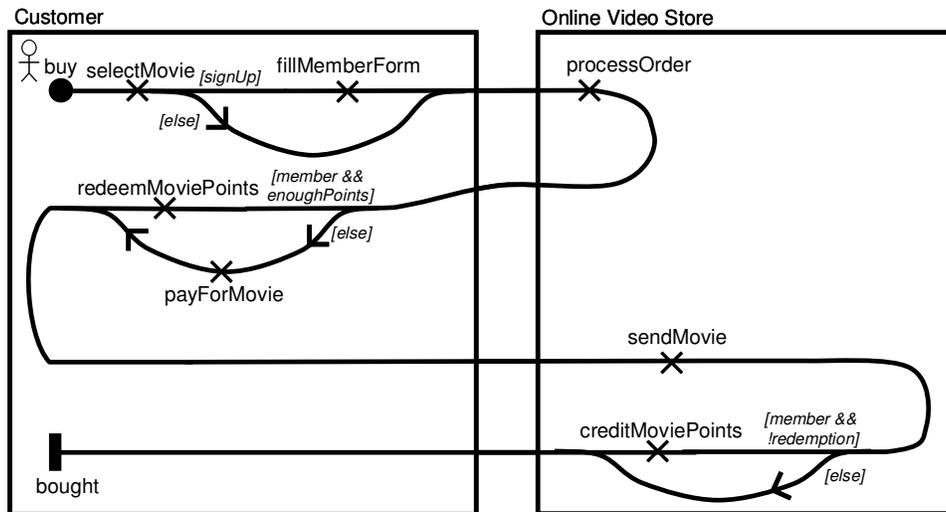


Figure 89 Combined Order Movie and Movie Points Scenarios

With AoUCM, however, the overall behavior of each concern can still be shown on its aspect map with one easily-understood, uninterrupted path without losing contextual information. Figure 90 shows the complete scenario model for the Movie Points concern. The three pointcut stubs represent where the two scenarios are interleaved, i.e., at the processOrder, payForMovie, and sendMovie steps. The aspectual behavior of the Movie Points concern is arranged around the pointcut stubs, stating that the fillMemberForm behavior occurs before processOrder, redeemMoviePoints is an alternative to payFor-

Movie, and creditMoviePoints happens after sendMovie. Consequently, the combined behavior is modeled exactly as desired while the causal relationships between the responsibilities of the Movie Points concern are still clearly expressed even with multiple pointcut stubs added to the aspect map.

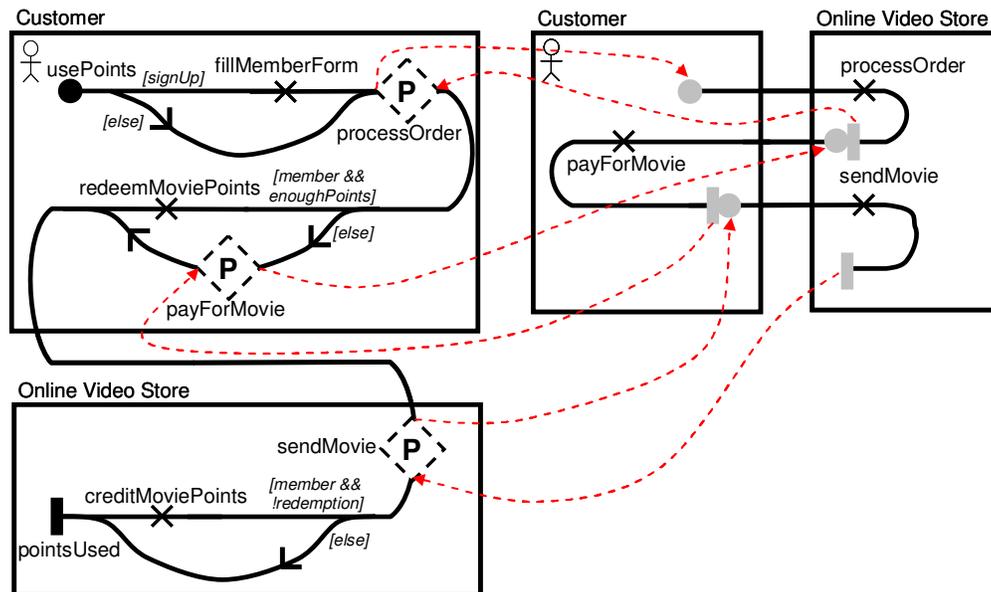


Figure 90 Modeling the Movie Points Concern

All pointcut stubs have the pointcut map on the right as their plug-in map. The plug-in bindings of the pointcut stubs formally define the interleaved composition with the help of additional grey end point/start point pairs on the pointcut map. The pointcut map is still only one single pattern that is matched in its entirety against the base model, i.e., the pattern processOrder, payForMovie, and sendMovie. Grey start points and grey end points are not part of the matching pattern even if they are connected. They are simply ignored by AoURN's matching mechanism. The additional end point/start point pairs, however, are used by AoURN's composition mechanism to compose aspectual behavior relative to a segment of the pattern.

Finally, the AoViews of the Movie Points concern can be shown just like for any other concern. Figure 91 shows a compact version of the AoViews of the Movie Points concern with the help of multiple colors. By following the plug-in bindings of the aspect markers, the resulting composed behavior is the same as the desired composed behavior described earlier.

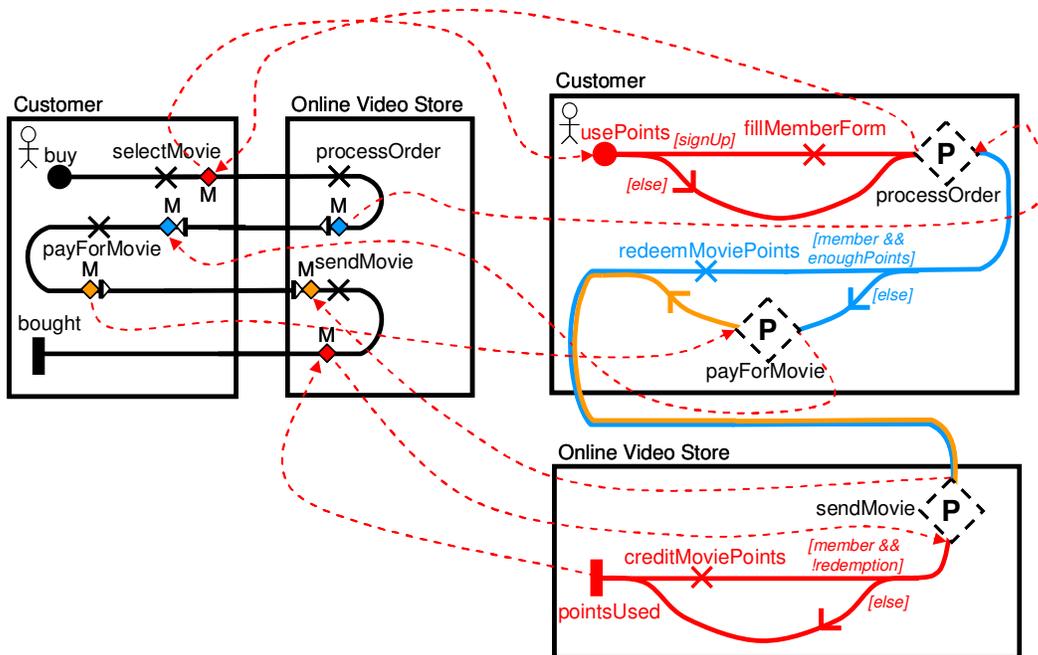


Figure 91 AoViews of Movie Points Concern Applied to OVS System

6.6. AoURN Metamodel

The concrete syntax of URN does not need to be altered significantly, because (a) the concern interaction graph, aspect diagrams, and pointcut diagrams are syntactically the same as traditional URN diagrams, (b) the tagging approach used for the concern interaction graph and AoGRL is already supported by URN metadata, (c) AoUCM aspect and pointcut stubs including all their variations are just specializations of static and dynamic stubs, (d) local start and end points are already supported by UCM 2.0, (e) the start and end of pointcut expressions are just unnamed start and end points, and (f) variables, wild-cards, and logical operators can already be specified with the existing naming facilities for URN model elements. Furthermore, the concern concept does not have its own visual representation. Only the anything pointcut element is a new path node element with a completely new concrete syntax. Therefore, requirements engineers can continue working with the familiar UCM notation as before. The abstract syntax, however, does need to be changed, but the changes are rather small.

6.6.1 Summary of Concrete Syntax of AoURN

Figure 92 gives a brief description of all types of AoURN diagrams, i.e., concern interaction graph, aspect graph, pointcut graph, aspect map, and pointcut map. Furthermore, all AoGRL and AoUCM modeling elements are shown.

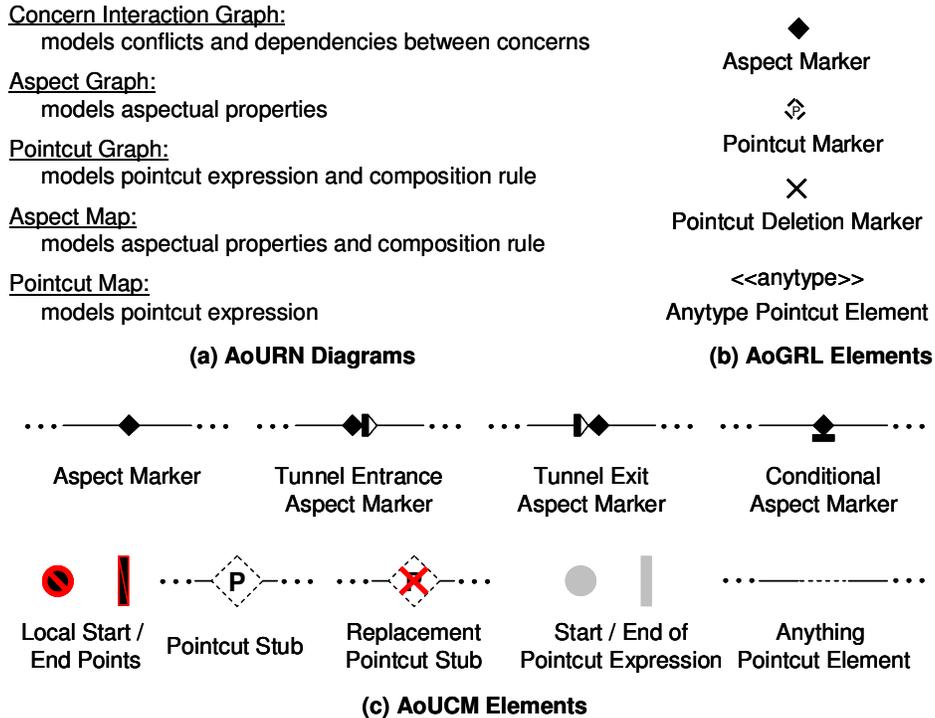


Figure 92 Summary of AoURN Elements

For AoGRL, pointcut markers, pointcut deletion markers, and the anytype pointcut element are used on pointcut graphs while aspect markers are automatically added to the GRL model. For AoUCM, pointcut stubs, pointcut replacement stubs, and local start/end points are used on aspect maps, while grey start/end points and the anything pointcut element are used on pointcut maps. Any kind of aspect marker may be added automatically to the UCM model.

6.6.2 Summary of Abstract Syntax of AoURN

Modeling concepts for AoURN can either be defined as extensions to the URN meta-model or through the specification of metadata that is attached to existing URN meta-model elements. AoURN strikes a balance between keeping the URN metamodel simple and the fundamental needs of an aspect-oriented requirements notation. AoURN's ap-

proach therefore differentiates between goal vs. scenario models and the specification of scenario concerns vs. the composition of scenario concerns. First, the fundamental concept of concern is added to the URN metamodel. It is even in the latest version of the URN standard. Second, anything related to aspect-oriented goal modeling is expressed with the help of metadata because AoGRL’s approach itself is a tagging approach. This metadata approach has no impact on the URN metamodel and keeps the aspect-oriented goal modeling simple. Third, anything related to the specification of scenario concerns is added to the URN metamodel, because new modeling elements are required and any aspect-oriented UCM notation is expected to need these fundamental elements. Fourth, basic scenario composition is supported directly with extensions to the URN metamodel, but features required by more advanced scenario composition are captured by metadata. This decision allows a particular incarnation of AoURN to support select advanced composition features without having to support various versions of the URN metamodel. Note that any URN model is also a valid AoURN model as the AoURN metamodel is an extension of the URN metamodel as shown in Figure 93.

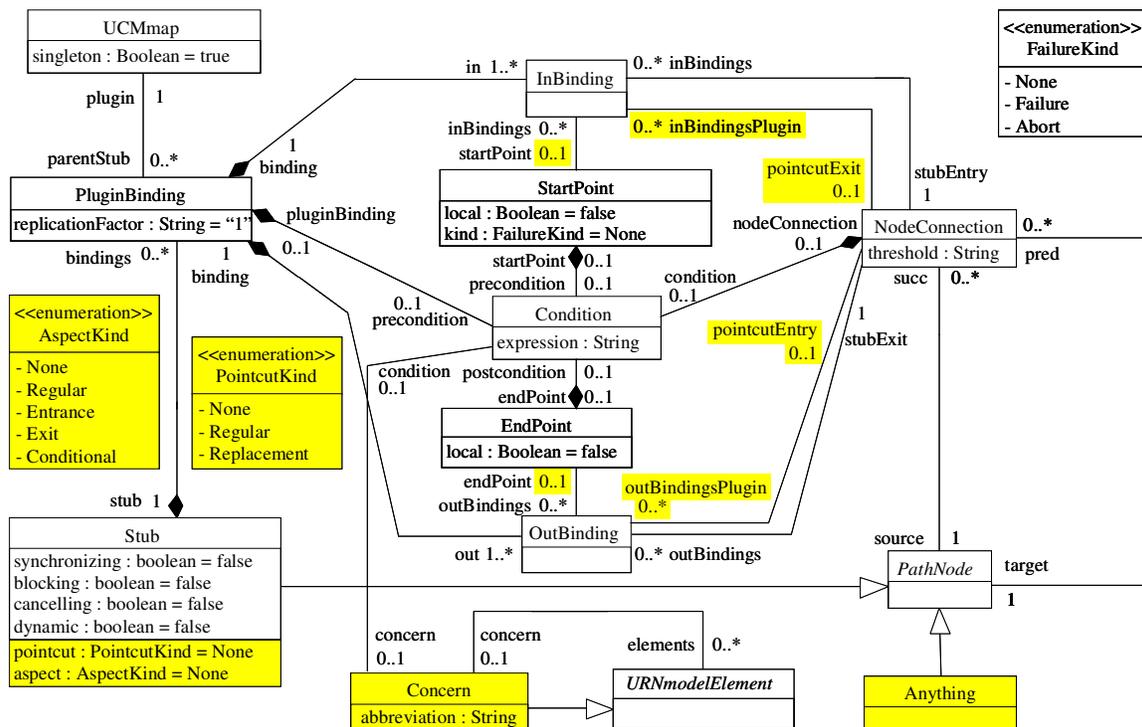


Figure 93 URN Metamodel with Extensions for AoURN

Figure 93 shows a simplified URN metamodel with highlighted extensions required for the support of aspect-oriented modeling. A Concern is a URNmodelElement and groups URNmodelElements that can be either UCMmaps, GRLgraphs, other Concerns, or IntentionalElements on the concern interaction graph. Concern has an abbreviation and may have a Condition indicating when the Concern is enabled. Conflicts, dependencies, and precedence rules among concerns are modeled with the concern interaction graph and are therefore captured by the existing GRL metamodel.

Anything is a new PathNode that represents the anything pointcut element. Constraints enforce that the anything pointcut element is connected to exactly one previous and one next path node, and that the anything pointcut element can be used only on pointcut maps. Dynamic stubs may also be characterized as pointcut stubs (see Figure 94 for a visualization of the dependencies between all nine types of stubs). A pointcut stub may be either a regular pointcut stub (Regular) or a replacement pointcut stub (Replacement). None indicates that the stub is not a pointcut stub. Static and dynamic stubs may now be further characterized as aspect stubs. An aspect stub may be one of four kinds: regular (Regular), tunnel entrance (Entrance), tunnel exit (Exit), and conditional (Conditional). None indicates that the stub is not an aspect stub. If the stub is an aspect stub, additional plug-in map bindings may be established, requiring optional associations between InBinding and StartPoint as well as OutBinding and EndPoint. The InBinding may now also connect the stub's in-path with a pointcut stub's out-path on the plug-in map (the pointcutExit NodeConnection) and the OutBinding may now also connect the stub's out-path with a pointcut stub's in-path on the plug-in map (the pointcutEntry NodeConnection). The concrete syntax for the aspect stub is the aspect marker in AoUCM models.

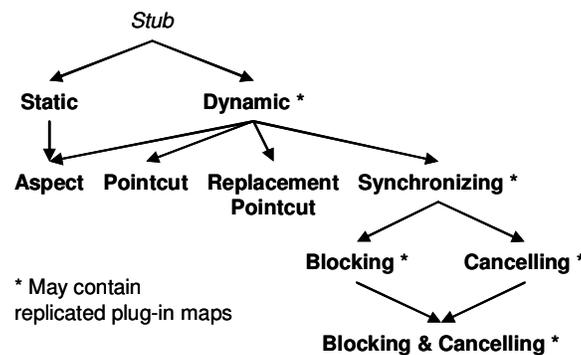


Figure 94 Types of Stubs

In addition to the changes to the URN metamodel, a number of metadata name/value pairs exist that are required for AoURN to enable the tagging of AoURN model elements. Figure 95 presents another excerpt from the URN metamodel, indicating what metadata is allowed for which metamodel elements. The tagging approach is mostly used for AoGRL. A GRLgraph with the metadata name/value pair `aspect/interaction` is a concern interaction graph. Unresolved conflicts among an `IntentionalElement` or `Actor` on the concern interaction graph are tagged with the metadata name/value pair `aspect/conflict N`. An `IntentionalElement` or `Actor` with the metadata name/value pair `aspect/aspect marker` relates to a node in a GRL graph that is transformed by an aspect.

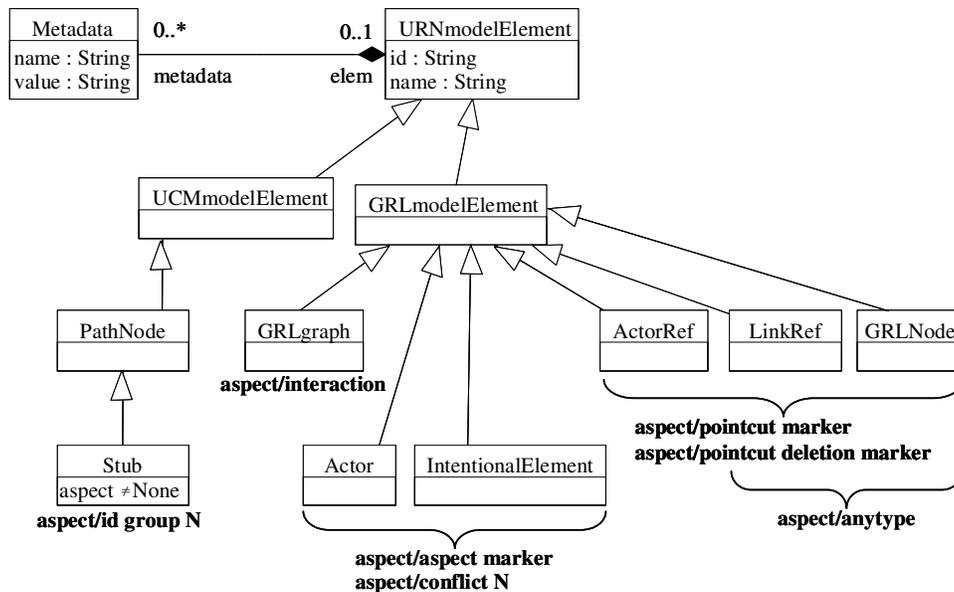


Figure 95 Metadata for AoURN

In contrast to aspect markers and conflicts, the remaining metadata name/value pairs for GRL elements operate on a GRL graph's references to model element definitions and not on the model element definitions directly. An `ActorRef`, a `GRLNode`, or a `LinkRef` with the metadata name/value pair `aspect/pointcut marker` is a GRL model element that is part of a pointcut expression on a pointcut graph and matched against the base model. An `ActorRef`, a `GRLNode`, or a `LinkRef` with the metadata name/value pair `aspect/pointcut deletion marker` is a GRL model element that is also part of a pointcut expression on a pointcut graph and matched against the base model but also removed by the aspect from the base model. Furthermore, a `GRLNode` tagged with `aspect/pointcut marker` or with `aspect/pointcut deletion marker` may also be tagged with the metadata name/value pair `as-`

pect/anytype to indicate that it may be matched to any kind of GRLNode. The same applies to LinkRef.

Some aspect-specific information is not reflected in the AoURN metamodel but encoded with the help of metadata. These are results of AoURN's matching and composition algorithms required for the visualization of AoViews. A Stub that is an aspect marker (i.e., the aspect attribute is not equal to None) may therefore be marked with the metadata name/value pair `aspect/id group N`.

The five types of AoURN diagrams shown in Figure 92.a are not reflected in the AoURN metamodel because concern interaction graphs are indicated by metadata and the remaining four diagram types are identified implicitly by their usage (i.e., an aspect map is a map with a pointcut stub, a pointcut map is bound to a pointcut stub, a pointcut graph contains an element with a pointcut marker or pointcut deletion marker, and an aspect graph is referenced in a pointcut graph).

Extensions to the Traversal Mechanism – While the static semantics is defined by the metamodel, the dynamic semantics of AoURN is defined by the GRL evaluation mechanism and the UCM path traversal mechanism. The GRL evaluation mechanism does not need to be changed for AoURN, because the evaluation mechanism is applied on the composed GRL model (which of course is a standard GRL model).

Several small changes are required for the traversal mechanism to understand AoUCM models. The traversal mechanism has to be able to traverse an aspect stub by continuing on to a start point or out-path of a pointcut stub on an aspect map and exiting the aspect map at an end point or in-path of a pointcut stub with the help of the plug-in bindings defined for the aspect stub. The general behavior, however, is identical to other stubs. The traversal mechanism relies on the composition process to establish all required plug-in bindings. In addition, the traversal mechanism must also pay attention to precedence rules specified on concern interaction graphs when traversing aspect stub with multiple aspect maps as plug-in maps. Furthermore, pointcut maps simply need to be ignored when traversing an AoUCM model.

The advanced features of AoUCM result in additional requirements for the improved traversal mechanism capable of understanding AoUCM models. First, the traversal mechanism must substitute variables with actual matched elements based on the

mappings established by the matching process. Second, the traversal mechanism must be able to enter a plug-in map through one aspect stub and exit the plug-in map through a different aspect stub. Appendix B: UCM 2.0 Traversal Mechanism provides detailed requirements for the improved traversal mechanism for UCM 2.0.

6.7. Examples of AoURN Models

This section discusses several additional examples of AoURN models including a reservation system, an online news system, and an electronic voting system. A fourth example, the YKeyK system, can be found in Section 6.8. The reservation system illustrates the advantages of using AoUCM over traditional ways of structuring UCM models such as the all-in-one, stubs, and root maps approaches. The online news system makes use of most major AoURN composition rules (i.e., insertion of GRL elements but not deletion of GRL base elements; sequential/concurrent/loop/interleaved composition but not replacement for UCM). The electronic voting system covers replacement for UCM as well as the following advanced AoUCM features: tunnel aspect markers, the anything pointcut element, as well as variables for the reuse of behavioral and structural base elements.

6.7.1 AoUCM Model for Reservation System

The advantages of aspect-oriented modeling of scenarios are illustrated with the help of a reservation system. Given the following requirements, the basic UCM model in Figure 96 can be created consisting of two use cases and one non-functional requirement:

1. System shall allow the user to make a reservation.
2. System shall allow the user to cancel a reservation.
3. System shall authenticate users before making or cancelling a reservation.
4. System shall add failed reservation attempts to a waiting list from which they will be taken when another reservation is cancelled in the future.

In the **Make Reservation** use case, the system first checks for authorization and only if the user is authorized is the reservation made. If the reservation is not successful, the reservation is added to a waiting list. In the **Cancel Reservation** use case, the system again checks for authorization and only if the user is authorized is the cancellation performed. After the cancellation, the system tries to fulfill a reservation from the waiting list. Con-

sidering that there exist other non-functional requirements in addition to security that need to be added to the use cases and considering that there exist other features that interact with making and cancelling reservations in addition to waiting lists and that need to be added to the use cases, there is clearly a need for better modularization. A more advanced UCM model which extracts the common behavior into a Security plug-in map is shown in Figure 97.

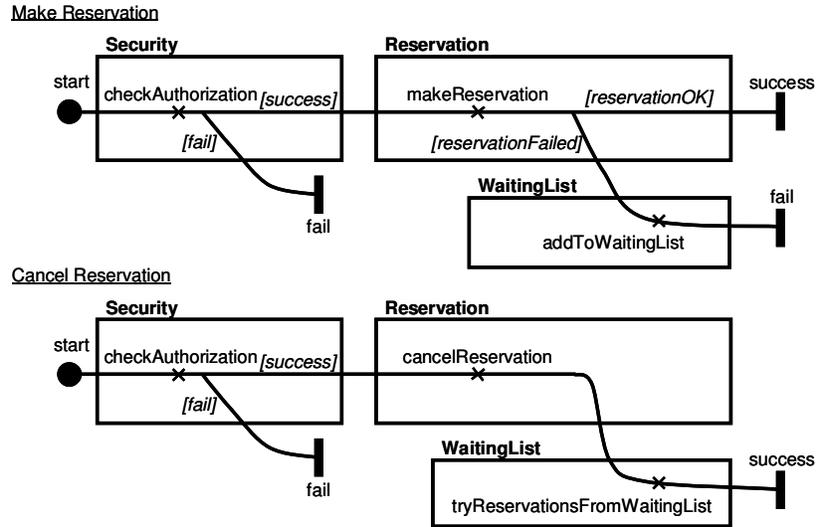


Figure 96 Basic UCM Model for Reservation Use Cases

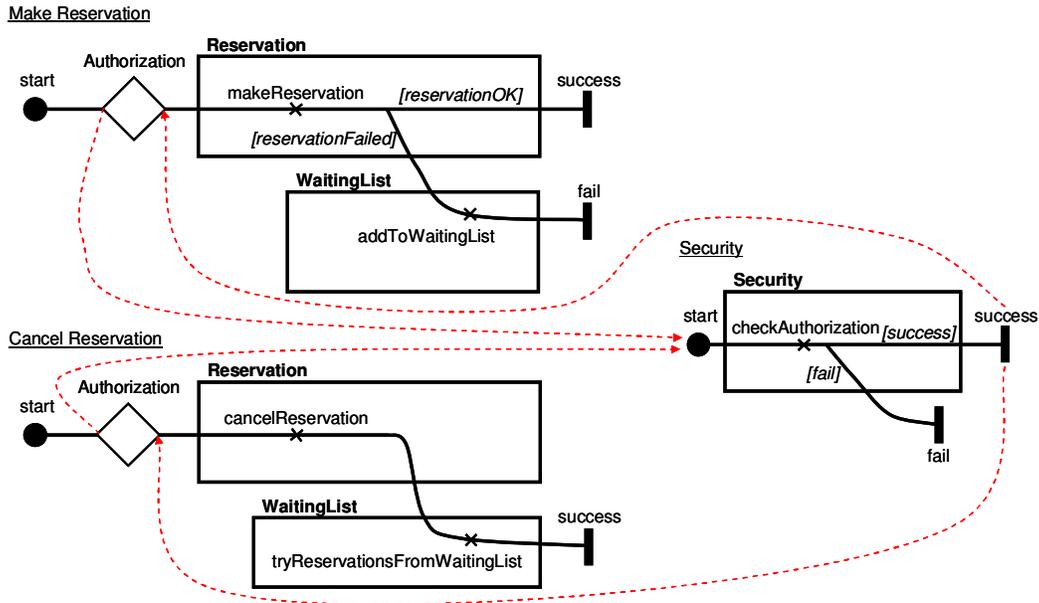


Figure 97 Advanced UCM Model for Reservation Use Cases – First Attempt

Even though the Authorization stubs structure the UCM model in Figure 97 considerably better, the Cancel Reservation and Make Reservation use cases still include descriptions of non-related behavior, i.e., stubs related to authorization and responsibilities related to waiting lists. Finding non-related behavior is not an easy task as this relates to concern identification. Heuristics can often be used such as looking for optional behavior that may have to be plugged into the system or for behavior that could possibly benefit from being considered a feature. In this case, Authorization is chosen as it relates to the well-established security concern. The waiting list behavior is chosen because it is optional for the reservation scenario as it may be required only for some reservation systems. Even though the waiting list behavior is not expected to be as crosscutting as the authorization behavior, it is a feature that could be plugged into a system as needed. The waiting list behavior exemplifies concerns that are not highly crosscutting, but still can benefit from aspect-oriented modularization techniques, because they are still important units of interest to the requirements engineer that may have to be managed separately from the rest of the system.

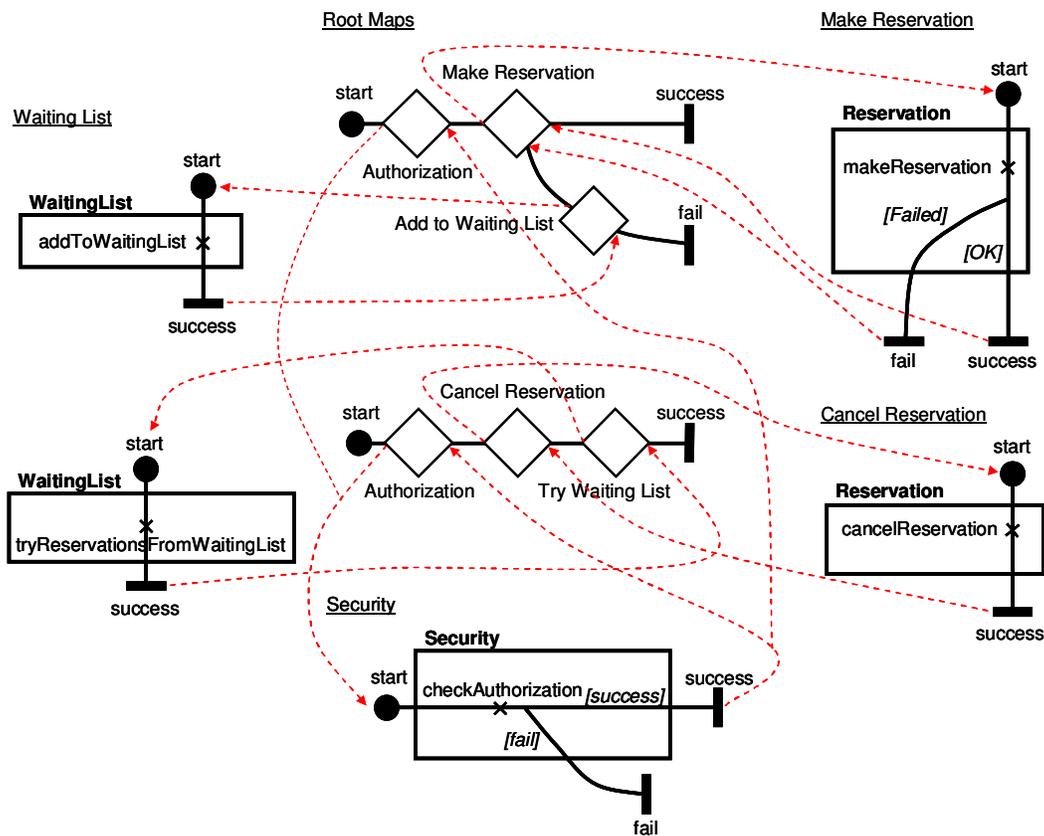


Figure 98 Advanced UCM Model for Reservation Use Cases – Second Attempt

A further improvement of the UCM model in Figure 97 is shown in Figure 98. Now, each individual plug-in map deals with only one concern. Top-level maps (often called Root Maps) describe how the different concerns are composed together. However, root maps often turn out to be rather complicated. Scalability is also an issue as, for example, the Authorization stub has to be added explicitly to each root map. Furthermore, the use cases are described by one plug-in map each only because no use case exists in this simple example that needs to be interleaved with another use case, causing the first use case to be split up into several disjoint plug-in maps. These disjoint maps make it much harder to understand and maintain individual use cases. AoUCM addresses these problems.

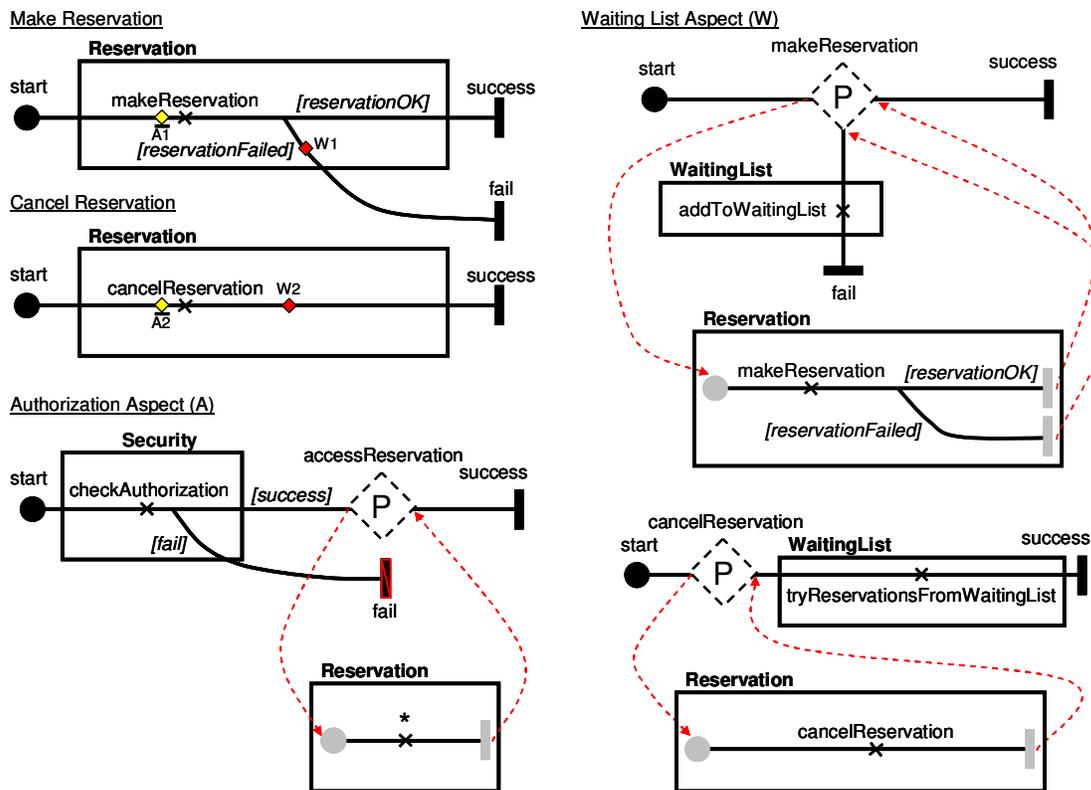


Figure 99 AoUCM Model for Reservation Use Cases

In Figure 99, the reservation system is modeled with AoUCM and all concerns are now fully encapsulated. Two concerns encapsulate the use cases for making and cancelling a reservation. Two aspects are defined that describe authorization and waiting list behavior, respectively. The Authorization aspect contains one aspect map and one pointcut map. For the Waiting List aspect, two aspect maps and two pointcut maps are defined. The authorization aspect adds an authorization check (`checkAuthorization`) before any responsi-

bility in the Reservation component, while the waiting list aspect adds failed reservations to a waiting list (`addToWaitingList`) and tries to fulfill a reservation from the waiting list if a cancellation occurs (`tryReservationsFromWaitingList`). The aspect markers added because of the Authorization aspect are labeled with A1 and A2, while the aspect markers for the Waiting List aspect are labeled W1 and W2.

The pointcut map for authorization matches any access to the Reservation component. Therefore, the responsibility in the authorization pointcut map is matched against `makeReservation` and `cancelReservation` in the use case concerns. Note that it is irrelevant for the match that the make reservation map contains two end points and the cancel reservation map only one because the end points (as well as the start points and the OR-fork) are not matched. The pointcut map requires only the responsibilities to be matched.

The pointcut maps for the waiting list match the `makeReservation` responsibility in the first pointcut map with the `makeReservation` responsibility in the use case concern, the OR-fork including conditions in the first pointcut map with the OR-fork and conditions in the use case concern, and the `cancelReservation` responsibility in the second pointcut map with the `cancelReservation` responsibility in the use case concern.

6.7.2 AoURN Model for Online News System

The AoURN model in Figure 100 to Figure 102 is based on the following informal description of an online news system. In general, the model consists of two stakeholder concerns (Reporter and Webmaster), the Security concern, and several use case concerns.

1. The reporter wants to receive international recognition of her work. The reporter meets with informants, researches the story online and in the library, and takes notes after doing so. Informants are paid for their services. A payment is negotiated first. An initial sum is then paid before the meeting with the informant and the rest after the meeting.
2. The reporter writes the story after researching it. The story is shelved if it does not make the cut for publication but may be continued at a later point.
3. When a story makes the cut, it is published on the online news system's website by the webmaster. The webmaster wants to provide the content as quickly

as possible. If the story is very important, it is also added to the headline section of the website and to the RSS feed.

- Security is of concern to both, the reporter and the webmaster. While the reporter uses fingerprint technology to access his laptop and desktop computers, the webmaster is issued cardkeys for access control to the servers.

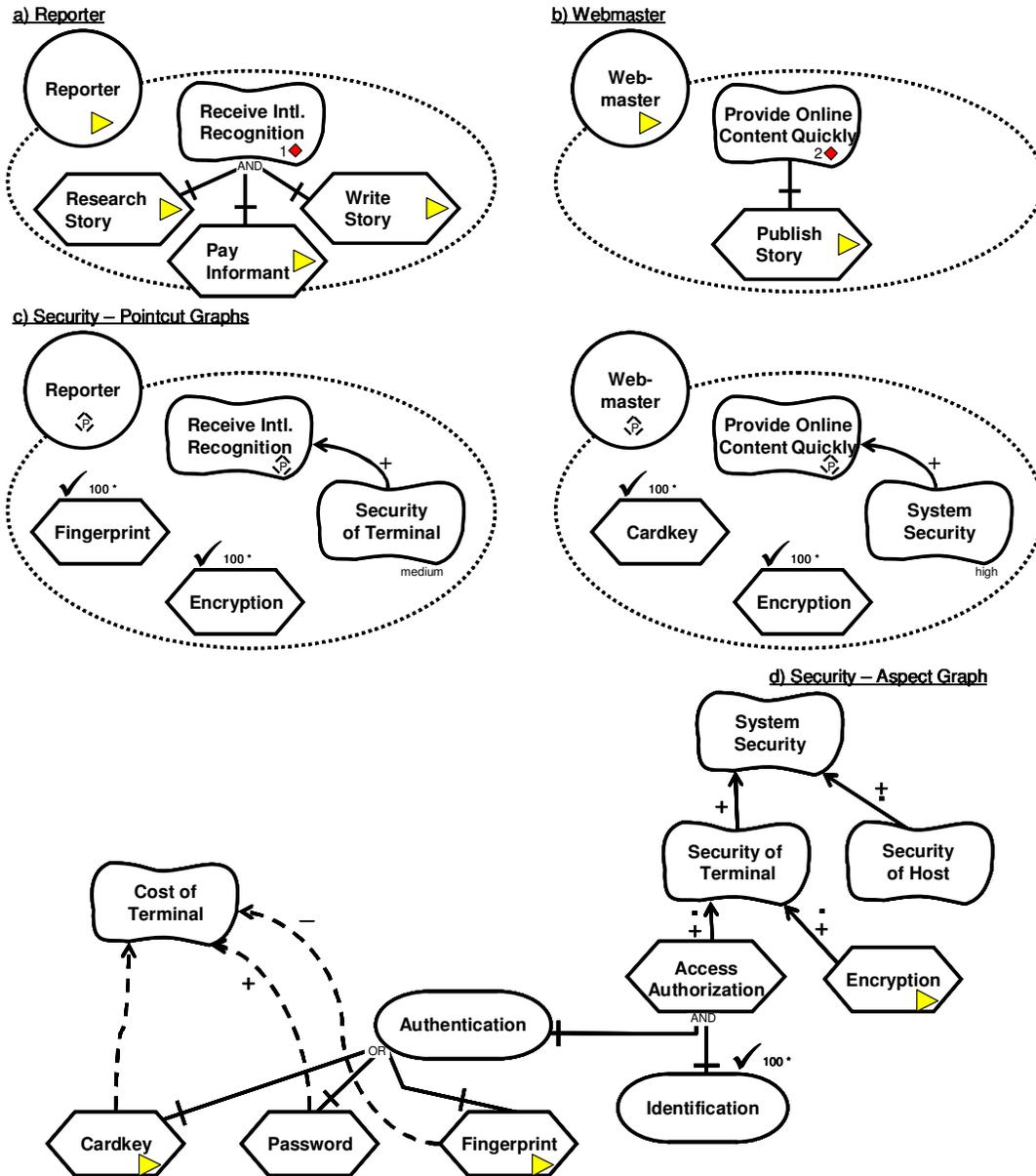


Figure 100 AoGRL Model for Online News System

The AoGRL model (Figure 100) describes the major tasks of each stakeholder and security implications with three concerns. For the AoUCM model (Figure 101 and Figure

102), six use case concerns can be identified in the description: Research Story, Pay Informant, Write Story, Shelve Story, Publish Story, and Publish To RSS Feed. Not all of these are crosscutting concerns but each is encapsulated in the AoUCM model to increase modularity. Three more crosscutting concerns are related to the Security concern in the AoGRL model: Authentication (Fingerprint), Authentication (Cardkey), and Encryption.

The Security aspect in the AoGRL model inserts security-specific GRL elements in each stakeholder goal graph. In the AoUCM model, Write Story and Publish Story are composed in sequence after Research Story and Write Story, respectively. Research Story and Pay Informant are an example for interleaved composition. The aspect markers of Pay Informant therefore belong to the same aspect marker group. Write Story and Shelve Story are composed with each other in a loop. The aspect markers of Shelve Story therefore belong to the same aspect marker group. Publish to RSS Feed is an example of a composition that requires concurrency. Authentication (Fingerprint), Authentication (Cardkey), and Encryption are further examples of sequential composition.

The pointcut maps of the Research Story and Write Story pointcut stubs in Figure 101 are not shown because the Research Story and Write Story maps, respectively, are reused for that purpose. For this particular sequential composition where complete scenarios are added once one after the other (i.e., first Research Story, then Write Story, and then Publish Story), more traditional UCM modeling techniques such as a) root maps or b) preconditions for starting points could also be used effectively. The root map would consist of three static stubs in sequence with Research Story, Write Story, and Publish Story as plug-in maps. Alternatively, preconditions such as researched and written could be added to the start points of the Write Story and Publish Story maps, respectively. Note, however, that only simple sequential ordering can be modeled this way. For the relationships expressed with the remaining pointcut stubs in Figure 101 and Figure 102, preconditions are not appropriate and root maps would become very complex.

Finally, there are conflicts between concerns because the Authentication (Fingerprint) concern is applied at the same location as the Shelve Story concern and the Authentication (Cardkey) concern is applied at the same location as the Encryption concern. To resolve the conflicts, the Authentication concerns must be applied before the other two concerns.

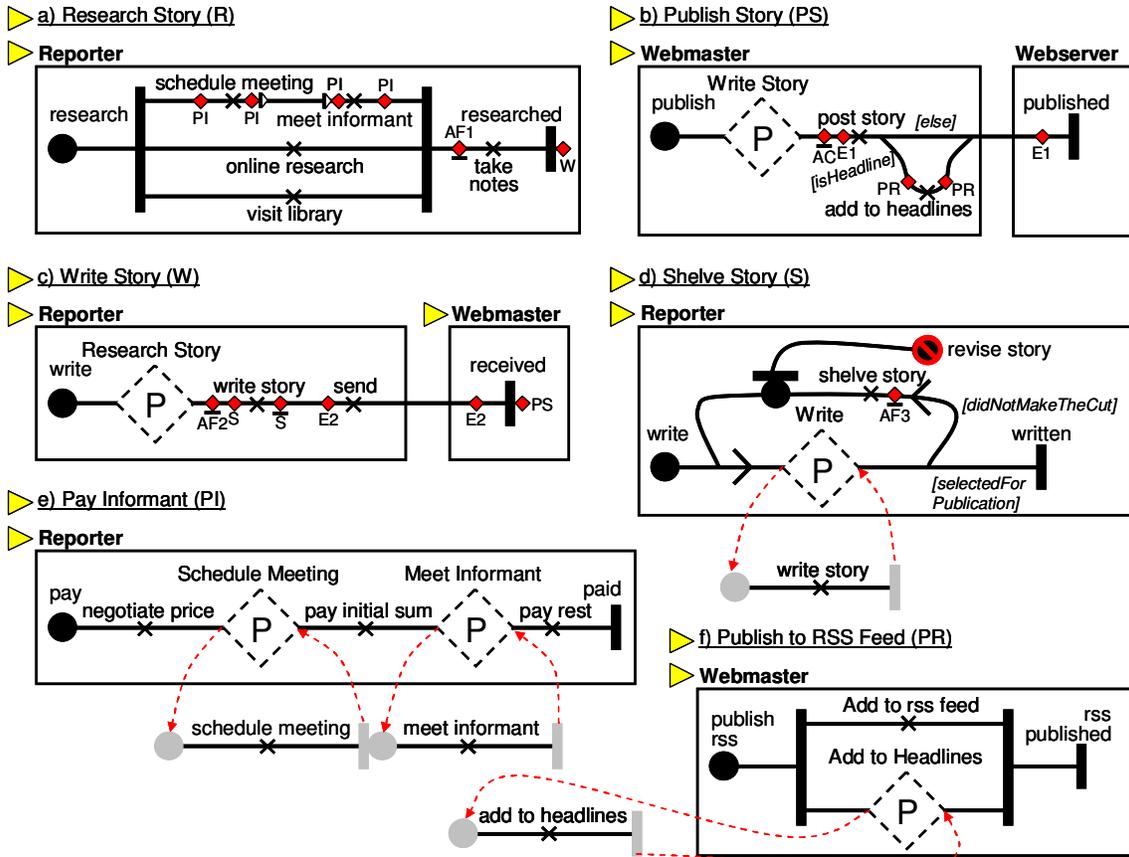


Figure 101 AoUCM Model for Online News System – Part I

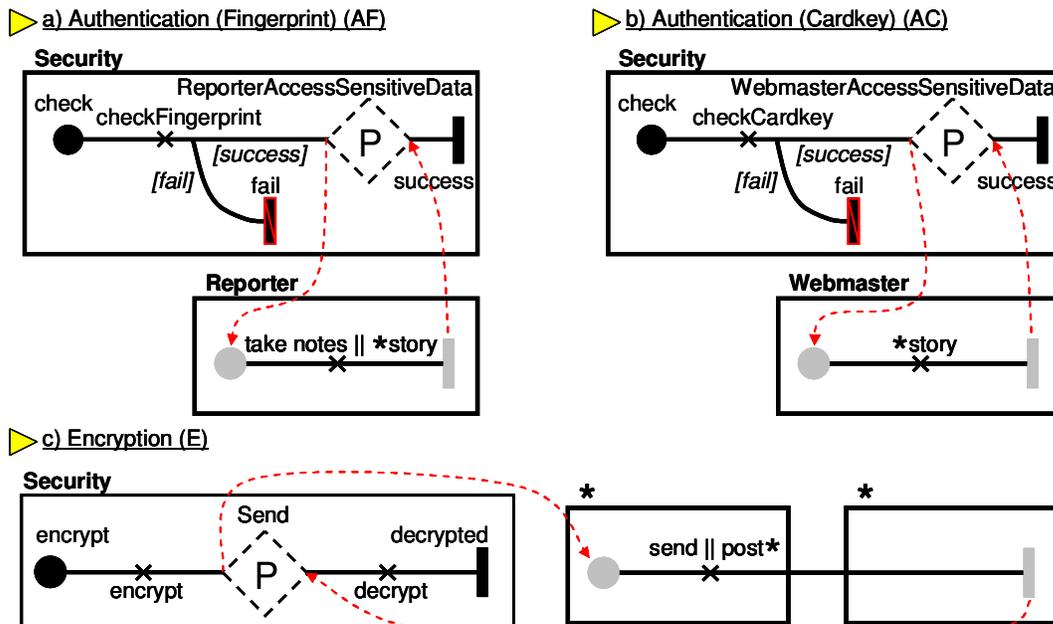


Figure 102 AoUCM Model for Online News System – Part II

The online news system shows that, even in simple systems, more complex composition rules beyond the usual before/after/around constructs are needed to model all causal relationships between concerns. An exhaustive composition technique such as AoURN's that is not limited by a particular pointcut expression language is therefore crucial to cope with the intrinsically complex relationships that exist between concerns in real-world systems.

6.7.3 AoUCM Model for Electronic Voting System

One of the principal use cases of an electronic voting system is presented in one single map in Figure 103 based on a description of Diebold's Electronic Voting System (EVS) [73]. In the Reporting use case, a responsible Poll Official interacts with the system to transmit the voting results to a central backend server. While the UCM model combines the use case concern with security (Authentication), distribution (Remote Service), and performance (Caching) concerns, the AoUCM model in Figure 104 encapsulates each of these concerns in its own model.

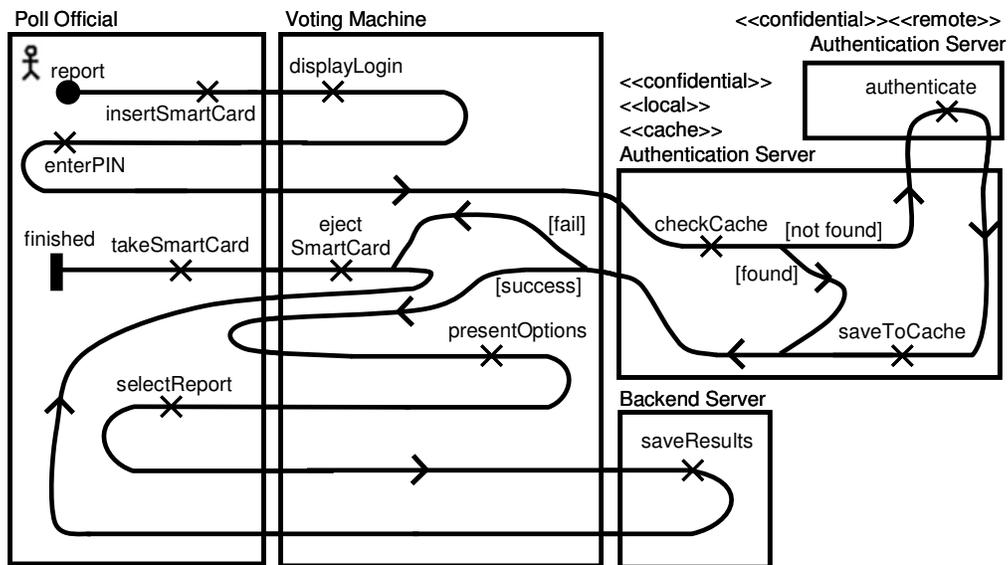


Figure 103 UCM Model for Reporting Use Case of Electronic Voting System

The Reporting concern in Figure 104 describes the principal use case of the system. The Authentication aspect introduces an authentication check involving a smart card before a scenario requiring authentication and ejects the smart card at the end of the scenario or when the authentication fails. The pointcut map of the Authentication aspect matches

against a path in the base model consisting of the `presentOptions` responsibility in the Voting Machine component, followed by a path of unspecified length (as indicated by the anything pointcut element), and finally the reported end point in the Poll Official component. When the Authentication aspect is applied, a conditional aspect marker is added after the report start point and another aspect marker is added after the reported end point. The conditional aspect marker is added because an OR-fork in the Authentication aspect may cause the pointcut stub to be circumvented and therefore the base behavior to be replaced with aspectual behavior. Both aspect markers belong to the same aspect marker group.

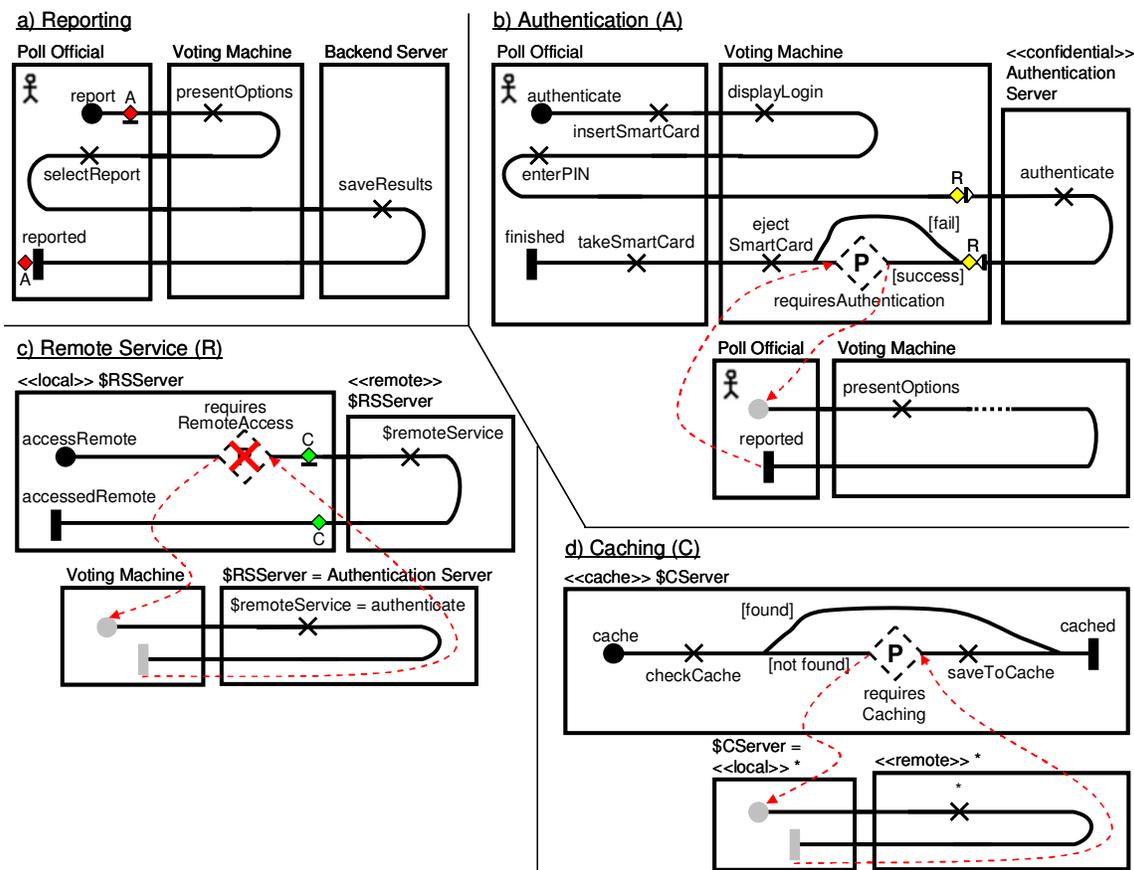


Figure 104 AoUCM Model for Reporting Use Case of Electronic Voting System

The combined behavior unfolds as follows: After entering the conditional aspect marker A, the scenario continues at the `authenticate` start point on the aspect map. If authentication is successful, the pointcut stub is reached and the plug-in binding to the conditional aspect marker A allows the scenario to exit the conditional aspect marker and continue

with the base scenario. When the second aspect marker A is reached on the Reporting map, the scenario continues at the exit of the pointcut stub and continues until the finished end point is reached. If, however, authentication is not successful, the [fail] path is taken, the pointcut stub is circumvented and the scenario reaches the finish end point. A plug-in binding exists from the finish end point to the second aspect marker A. Since the two aspect markers belong to the same aspect group, the scenario is allowed to continue along the plug-in binding and exits the second aspect marker A which is also the end of the scenario.

The Remote Service aspect models generically the provisioning of remote access without specifying what the remoteService, the <<local>> server, and <<remote>> server are. These generic concepts are represented by variables and will therefore be substituted by the matched elements. Since the Authentication Server corresponds both to the local and remote server, the original Authentication Server is split into local and remote parts. The <<local>> server forwards the request to the <<remote>> server. Furthermore, the usage of the replacement pointcut stub indicates that the matched base elements will be removed.

The pointcut expression of the Remote Service aspect matches against the authenticate responsibility in the Authentication aspect. One tunnel entrance aspect marker and one tunnel exit aspect marker are therefore added to the Authentication map. They belong to the same aspect marker group. The combined behavior therefore is as follows: When the tunnel entrance aspect marker is reached in the Authentication aspect, the scenario continues at the out-path of the requiresRemoteAccess pointcut stub, forwards the request, follows the path into the <<remote>> server where the substituted authenticate responsibility is executed before returning back to the <<local>> server. After reaching the accessedRemote end point, its plug-in binding to the tunnel exit aspect marker is followed. Effectively, the original authenticate responsibility in the Authentication aspect is skipped and moved from the Authentication Server in the Authentication aspect to the Remote Server. Finally, the metadata <<local>> and <<remote>> is applied to the matched component, in this case the Authentication Server from the Authentication aspect.

The Caching aspect models in a generic way the behavior of a cache. Its pointcut expression makes use of metadata to narrow down the matched elements to interactions between <<local>> and <<remote>> components. Similarly to the Authentication aspect, the Caching aspect contains an OR-fork. One conditional aspect marker and another aspect marker are therefore added to the Remote Service map when the aspect is applied. The two aspect markers belong to the same aspect marker group. The combined behavior unfolds as follows: When the conditional aspect marker is reached in the Remote Service aspect, the scenario continues at the cache start point. If the requested data is [found] in the cache, the pointcut stub is circumvented, and the cached end point is reached. Its plug-in binding is followed and the scenario continues at the second aspect marker without ever entering the <<remote>> server. If the data is [not found], the conditional tunnel aspect marker is exited and the scenario continues to the <<remote>> server. When the second aspect marker is reached, the scenario continues at the exit of the pointcut stub and the data is therefore saved to cache. Finally, the Caching aspect adds <<cache>> to the matched component because a variable is specified, i.e., in this case the <<local>> server in the Remote Service aspect.

The resulting composed system with all three aspects is semantically identical to the UCM model from Figure 103. It is also structurally equivalent after the aspect markers are transformed into a flattened UCM model.

6.8. Relationship of Goal and Scenario Concerns

AoURN's ability to encapsulate non-functional requirements (NFRs) as well as use cases in both model types helps bridge the gap between goal and scenario models. This gap is further narrowed by URN traceability links between modeling elements of goal and scenario concerns. The YKeyK model in this section exemplifies the relationship between AoURN goal and scenario concerns. YKeyK stands for Your Key Knows, a system that allows drivers to find their car in a car park by following directions shown on a small display of the car key [2].

Figure 105 shows the goal model of a part of the YKeyK system, described in a traditional way without aspects. The goal model defines the relationships between three major stakeholders, the Driver, the YKeyK system, and the Car Park. For each of the

stakeholder, softgoals model the general goals at the top of the goal graph. The Driver wants to quickly find a parked car, YKeyK wants to make it easier for the driver to find the car, and the Car Park wants to earn more money. These are refined into goals and eventually into tasks that correspond to use cases or parts of use cases (prefixed in this case with UC002, which stands for Use Case #2). Alternatives to the use cases are also documented – the Driver may rely on memory and the Car Park may provide clear numbering and signage. As indicated by URN links, tasks are further detailed in the UCM scenario model. The URN links of stakeholders trace the stakeholders to components in the UCM scenario model. Many more stakeholders and NFRs as well as the Visit Car Park use case exist but the presented YKeyK model suffices for this example.

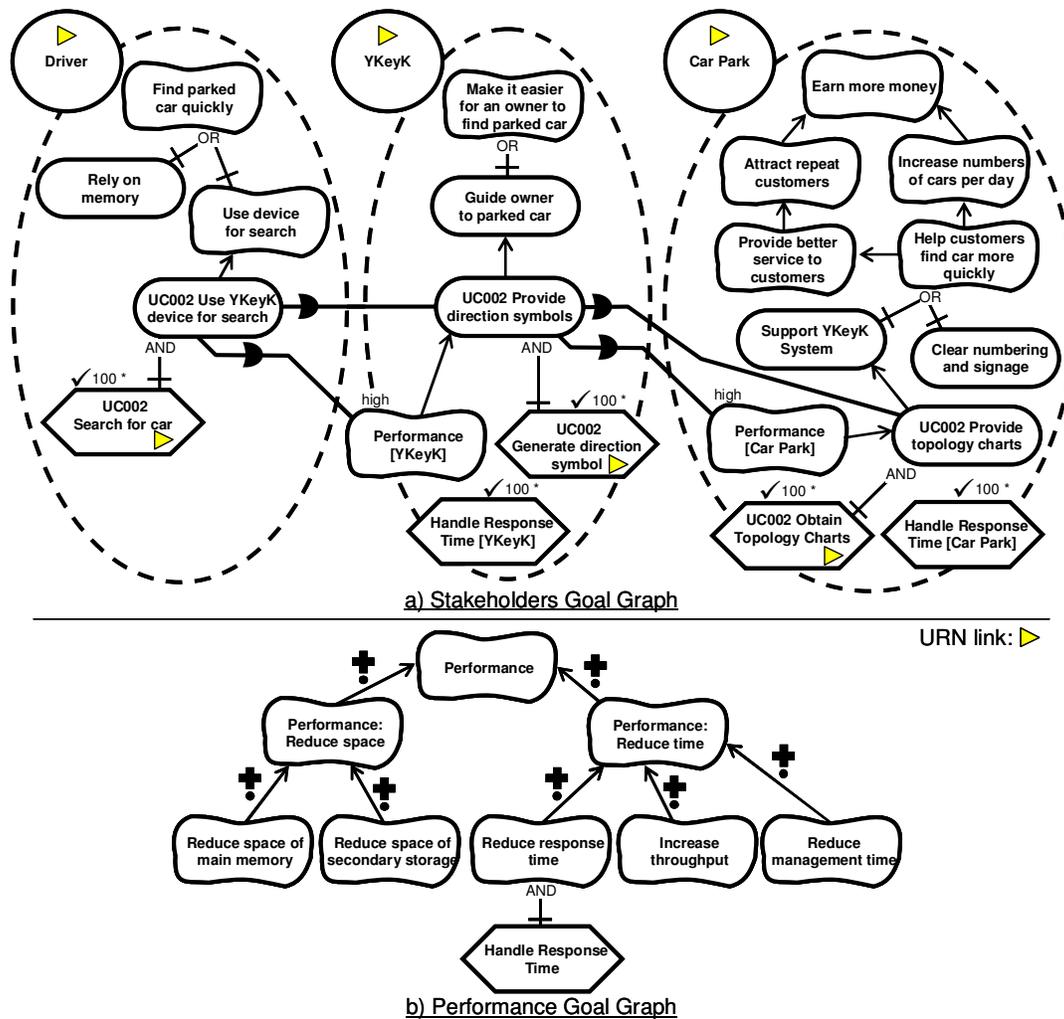


Figure 105 Goal Model of YKeyK System

The use case introduces dependencies between the stakeholders as the Driver depends on YKeyK to provide directions in order to search for the car with YKeyK. In turn, YKeyK depends on the Car Park to provide topology charts in order to be able to give correct directions. Furthermore, the Driver is concerned about Performance which introduces further dependencies on YKeyK and the Car Park. In order to address this concern, response time issues have to be handled by YKeyK and the Car Park as indicated by the task Handle Response Time.

In addition, the Performance NFR is modeled. The Performance NFR is a GRL catalogue and is reused for two stakeholders (i.e., a Performance softgoal exists in each stakeholder). Thus, the Performance NFR is crosscutting the stakeholders. Each Performance softgoal is parameterized to indicate that a different instance of the Performance goal graph exists for each of the stakeholders. This feature is not supported by the GRL notation or the jUCMNav tool. Instead, several distinct copies of the Performance goal graph have to be created. With AoURN, the Performance goal graph can be applied to each stakeholder without explicitly creating several copies. This maintenance improvement presents an advantage of using AoGRL models over traditional ways of structuring GRL models.

Often, use cases are not crosscutting in a scenario model but are rather peers to each other. When modeled in GRL, however, use cases are typically crosscutting several stakeholders because a use case often impacts the goals of many stakeholders as shown in Figure 105. With AoURN, crosscutting use cases can be properly encapsulated even in goal models, but this encapsulation can often be achieved with standard GRL techniques, too.

Figure 106 to Figure 109 show the aspect-oriented version of the YKeyK model. The AoURN model contains three concerns, one each for the Driver, YKeyK, and Car Park stakeholders, and three aspects, one for the use case UC001 Visit Car Park, one for the use case UC002 Search Car, and one for the Performance NFR. A stakeholder's concerns (i.e., only the goals and alternatives related to a stakeholder) are modeled separately for each stakeholder on a goal graph (Figure 106.a to c). The aspect for use case UC001 Visit Car Park consists of two maps (Figure 107 and Figure 108) and a pointcut graph which is not shown as it is similar in structure to the pointcut graph of the second use

case. The aspect for use case UC002 Search Car contains the UC002 Search Car Pointcut Graph and seven maps (Figure 106.d and Figure 109). The Performance NFR aspect contains the Performance Pointcut Graph and the Performance Aspect Graph (Figure 106.e and f).

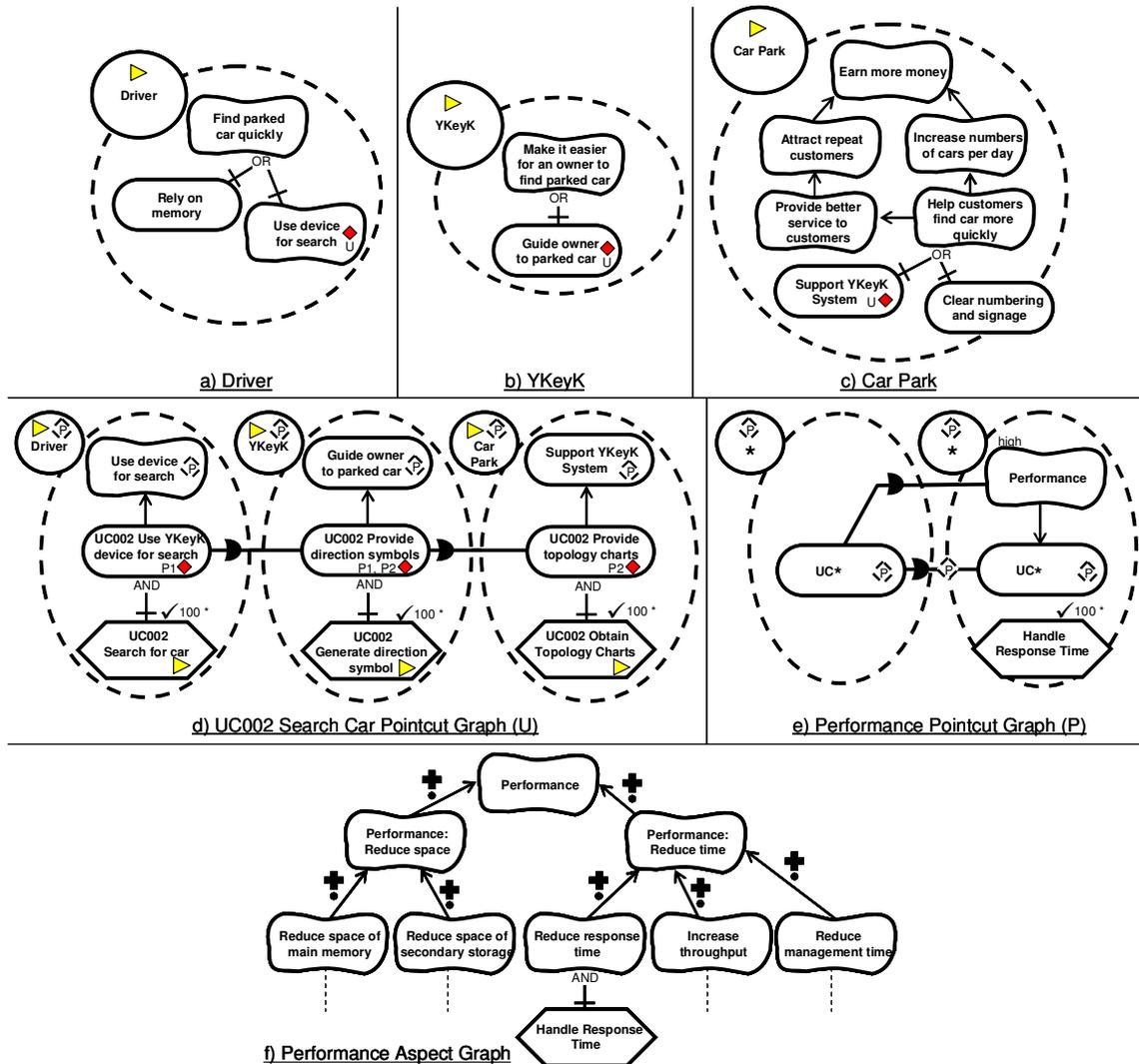


Figure 106 AoGRL Model of YKeyK System

The use case aspect in the goal model directly crosscuts the three stakeholder concerns while the Performance aspect crosscuts the stakeholder concerns via the use cases. In the goal model, the use case pointcut expression matches the three intentional elements in Figure 106.a to c as indicated by the aspect markers, thus adding the use case aspect to the stakeholder concerns. The Performance pointcut expression matches the two dependencies in Figure 106.d, thus adding the Performance softgoal and the Handle Response

Time task to the actor in which the target of the dependency resides (the YKeyK and Car Park actors in Figure 106.d). The Performance Aspect Graph is thus added to multiple stakeholders without explicitly duplicating the aspect graph. In addition, the Performance Pointcut Graph stipulates that a) a contribution to the target of the matched dependency and b) a dependency from the source of the matched dependency must be added. This pointcut graph essentially encodes the rule that all use cases of the YKeyK model (i.e., UC*) are concerned with performance and therefore any actor that depends on these use cases must also depend on performance issues.

Finally, URN links trace the Driver, Car Park, and YKeyK actors to their UCM components. The UC002 Obtain Topology Charts task in Figure 106.d is traced to path elements in the map in Figure 109.b as indicated. The UC002 Generate direction symbol task in Figure 106.d is traced to the generate direction symbol responsibility in the map in Figure 109.a, while the UC002 Search for car task in Figure 106.d is traced to each UC002 map in Figure 109.

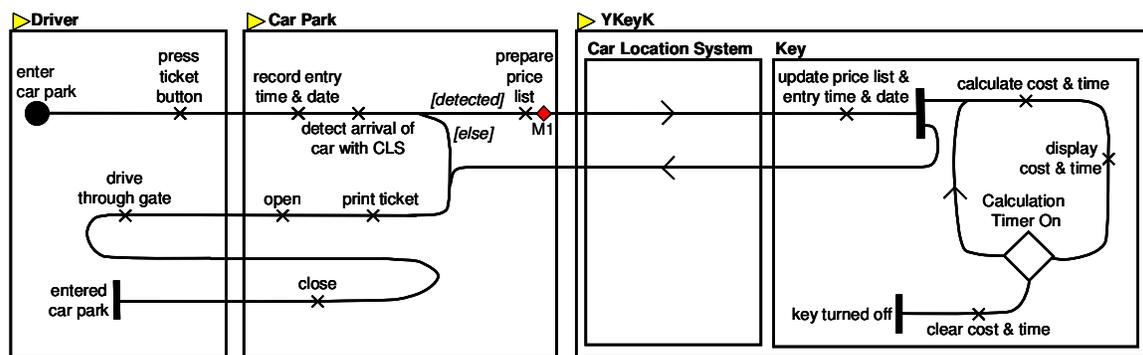


Figure 107 UC001 Visit Car Park – Enter Car Park Scenario of YKeyK System

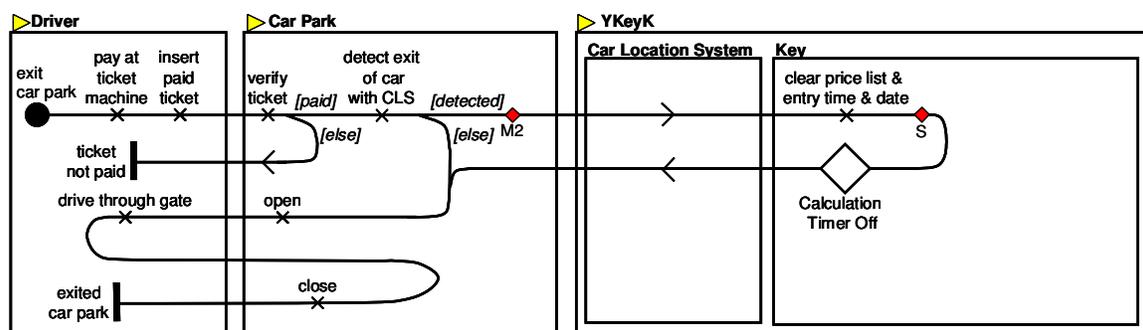


Figure 108 UC001 Visit Car Park – Exit Car Park Scenario of YKeyK System

The UC001 Visit Car Park aspect in the scenario model consists of two maps that describe the behavior for entering and exiting of the car park (Figure 107 and Figure 108). The UC002 Search Car aspect in the scenario model contains seven maps. Search for Car describes the main purpose of this aspect. The search, however, requires topology information about the car park which is transmitted when the car enters the car park. As this information is required only by the search capability but must take place during the execution of the UC001 Visit Car Park - Enter Car Park scenario (Figure 107), the required responsibilities are added with the help of an aspect (Figure 109.b) to the setup stage after the prepare price list responsibility in the Enter Car Park scenario. Furthermore, the signaling timer that is set by the main scenario of the Search Car use case when the search button is pressed needs to be turned off when exiting the car park in case the stop button has not been pressed. As this behavior is necessary only because of the search capability but must take place during the execution of the UC001 Visit Car Park - Exit Car Park scenario (Figure 108), the required responsibilities are added with the help of an aspect (Figure 109.c) to the exit stage after a car with the Car Location System (CLS) is detected in the Exit Car Park scenario.

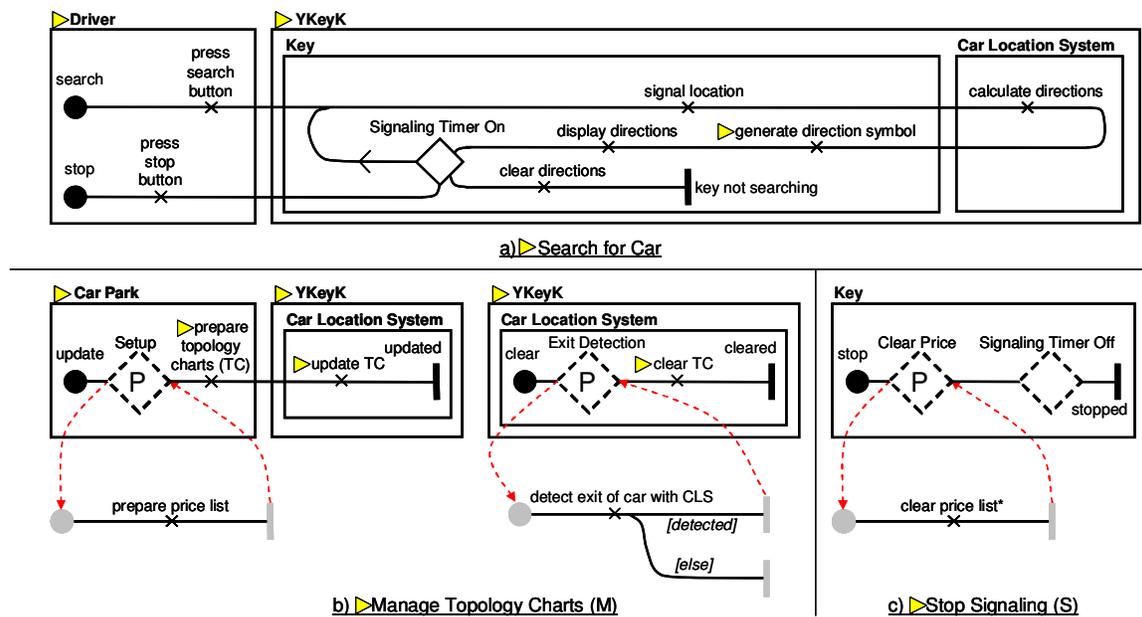


Figure 109 UC002 Search Car Use Case of YKeyK System

The use case aspects in the YKeyK example show that one concern in AoGRL may be traced to one concern in AoUCM. The use case aspects span the goal and scenario mod-

els. The performance aspect may also be traced to the scenario model. In this case, however, there are many ways of achieving performance improvements (e.g., caching, concurrency...) which are not elaborated in the YKeyK example. Each technique, however, should be modeled as a separate sub-concern of performance in AoUCM. Clearly, there is a one-to-many relationship between concerns in AoGRL and AoUCM.

Cases also exist where concerns in AoGRL are not traced to any concerns in AoUCM and vice versa. AoUCM models non-functional requirements that can be expressed or operationalized with scenarios. AoGRL, however, may model a much larger class of non-functional requirements. For example, the quality of a software product may be modeled in an AoGRL graph and it may be decided that inspections are the solution for this goal. This solution is not modeled in AoUCM (or UCM for that matter). Similarly, new concerns may appear in AoUCM models since there is a difference in abstraction levels of goal models and scenario models.

At this point, no evidence has been found for scenario concerns that relate to many goal concerns (Figure 110). However, in the spirit of “never say never”, this possibility is not categorically dismissed. In any case, if there is a relationship between concerns in goal and scenario models, AoURN can encapsulate these concerns across the two model types whether the concerns are crosscutting or not, and URN links keep track of these relationships.



Figure 110 Relationship of Goal and Scenario Concerns

6.9. Summary

This chapter introduces the Aspect-oriented User Requirements Notation (AoURN) with its two sub-notations, the Aspect-oriented and Goal-oriented Requirement Language (AoGRL) and Aspect-oriented Use Case Maps (AoUCM). The join point model of AoURN covers all semantically significant, behavioral and structural features of the notations, allowing aspects to transform any essential part of an AoURN model.

Each notation models aspects in its own, most appropriate way. AoGRL specifies aspectual properties on separate aspect graphs, while pointcut expressions and composition rules are both defined on pointcut graphs reflecting the highly interdependent and unstructured nature of GRL models. It is expected that aspect graphs are reused in a similar way as GRL catalogues, while pointcut graphs will not be reused as they tend to be very situation-specific with pointcut expressions and composition rules combined. AoUCM, on the other hand, takes advantage of the structuring mechanisms provided by the UCM notation and specifies aspectual properties and pointcut expressions on separate maps called aspect maps and pointcut maps, respectively. Composition rules are also defined on aspect maps which often are nevertheless an intrinsic part of the aspect and highly reusable. Pointcut expressions are encapsulated on pointcut maps and easily reused.

The concern interaction graph, an annotated but standard GRL graph, captures conflicts and dependencies that may exist between concerns, defines resolution mechanisms based on precedence rules, and also keeps track of any unresolved concern interactions.

AoGRL relies on metadata annotations of the GRL model to indicate where aspects are applied (aspect marker) and to identify the pointcut expression (pointcut marker, pointcut deletion marker, anytype pointcut element). The specification of aspectual properties and composition rules does not require extensions to GRL. AoUCM introduces new path nodes to indicate where aspects are applied (aspect marker – a specialization of a stub), for modeling aspectual properties (pointcut stub – a specialization of a dynamic stub), and for modeling the pointcut expression (the anything pointcut element – a new type of path node). Furthermore, unnamed start and end points on pointcut expressions carry the semantics of delimiters marking the beginning and end of pointcut expressions, respectively. The specification of composition rules, however, does not require extensions to UCM.

AoViews use the models specified by the requirements engineer to visualize the impact of applying concerns to the model and to navigate the composed system in an aspect-oriented way without the need to transform the composed system into a standard URN model. This approach avoids complex layout issues as the models to be visualized

are properly arranged by the requirements engineer during the specification of the concern.

More advanced features of AoURN are discussed in greater detail. These allow for more complex concerns to be specified and include a) the anything and anytype pointcut elements that allow variations in the pattern match, b) variables for the reuse of existing base behavior and structure, c) various types of aspect markers to characterize the impact of a concern on the base model, d) the usage of local start and end points on aspect maps, and e) interleaved composition rules that enable independent modeling of interleaved scenarios without losing contextual information important for maintaining an overall understanding of the scenarios.

All extensions to URN are formally defined in the AoURN metamodel, the usage of metadata is summarized, and the concrete syntax of AoURN modeling elements is illustrated.

Finally, the relationships between concerns in goal models and concerns in scenarios models are exemplified through the YKeyK example. AoURN allows concerns to be encapsulated across model types whether they are crosscutting or not. While use cases often do not crosscut each other in scenario models, they crosscut stakeholder concerns in goal models as uses cases tend to impact several stakeholders. Consequently, non-functional requirements that impact use cases also crosscut stakeholders via the use cases. It remains to be seen, however, whether aspect-oriented techniques in general are applicable to goal models at the very early stages of requirements engineering. During this stage, the needs and dependencies of stakeholders are typically captured in a highly unstructured, inconsistent model. For these cases, a simple tagging mechanism (which is supported by the AoURN metamodel) or the use of correlations may be more effective to identify and track concerns. The tagging mechanism essentially allows the goal model to be painted in different colors, each one belonging to a different concern. Correlations, on the other hand, allow impacts of one concern on other concerns to be identified on a goal graph and selectively displayed. In later stages, however, it is useful to organize goal models in an aspect-oriented manner, as the impact of solutions (i.e., scenarios) should be clearly separated from the rationales and decisions captured in the goal graphs of the

stakeholders. In these stages, aspect-oriented thinking provides important guidance in how to structure goal models.

This chapter covers the specification and visualization of AoURN models. The matching and composition mechanisms are described but in a rather informal and example-based way. The matching mechanism establishes the mappings between pointcut expressions and the base model. The composition mechanism adds aspect markers to the model and enables the aspect-oriented navigation of AoURN models with the help of AoViews as well as the composition of AoURN models into standard URN models. The next chapter explains in more detail the algorithms used for the matching and composition mechanisms.

Chapter 7. Composition of AoURN Models

This chapter discusses in greater detail the composition of aspects with the base model given a model defined in the Aspect-oriented User Requirements Notation (AoURN) as discussed in Chapter 6. A typical AoURN model consists of regular URN diagrams for non-crosscutting concerns, a concern interaction graph, and aspect diagrams as well as pointcut diagrams for crosscutting concerns.

The composition of AoURN models requires features to a) facilitate the matching of pointcut expressions against the AoURN model, b) indicate the insertion of aspectual properties with aspect markers, which enables support for model understanding and navigation in an aspect-oriented way with AoViews, c) indicate for an aspect which other concerns are being transformed by the aspect, d) keep track of changes to the matches of a concern's pointcut expression, and e) compose aspects and the base model into a complete view of the system as a traditional URN model.

Section 7.1 discusses the syntactic and semantics-based matching algorithm that establishes the mappings between elements in pointcut expressions and elements in the base model. Section 7.2 defines the composition algorithm that identifies the insertion points for aspectual properties with aspect markers to support the AoView or presents the system as a traditional URN model. Section 7.3 discusses the advanced features of interleaved composition rules and composition of semantics-based matching results. Finally, Section 7.4 gives a brief overview of how the remaining features are provided with the results of the matching and composition algorithms. A more detailed explanation of the matching algorithm is given in Appendix D: Matching Algorithm and a more detailed explanation of the composition algorithm is given in Appendix E: Composition Algorithm.

7.1. Matching Algorithm

Generally, the matching algorithm establishes a mapping from pointcut expressions to the base model. For AoURN, the *base model* for a concern is the complete AoURN model except for (a) the diagrams of the concern that is to be matched, (b) the diagrams of the concerns that are to be applied in the next waves as defined in the concern interaction graph, (c) all pointcut expressions, and (d) the concern interaction graph. The base model changes therefore from one concern to the next. A concern is not applied to itself to avoid potential infinite insertion loops. Furthermore, a concern obviously has access to its own elements; hence there is not a strong case for requiring aspect-oriented techniques within a concern. The matching algorithm is based on the syntax of pointcut expressions as explained in Section 7.1.1, but also takes semantics into account for AoURN models as detailed in Section 7.1.2. Semantic matching increases the resilience of the matching algorithm against refactoring of the AoURN model.

Figure 111 defines the terms required for the discussion of the properties of the matching algorithm. A *mapping* links exactly two URNmodelElements, i.e., one element from the base model (*baseElement*) to another element from the pointcut expression (*pointcutElement*). Base elements and pointcut elements may both appear in multiple mappings, since a pointcut expression may be matched multiple times in the base model and a base element may be matched by multiple pointcut expressions. The *condition* attribute of a mapping indicates whether the mapping is conditional depending on the values of variables, component plug-in bindings, and responsibility plug-in bindings in the base model.

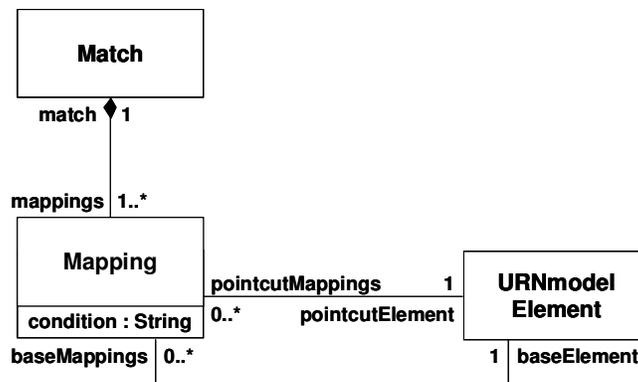


Figure 111 Domain Model for the Matching Algorithm

For AoGRL, the matching algorithm must operate on the definitions of base model elements, because GRL graphs are simply limited views of the complete base model and individually may not portray all relationships of the base model. The pointcut expression in AoGRL, on the other hand, is defined by the partial view of the GRL model presented in the pointcut graph. This GRL graph contains references to actors, intentional elements, and links. Therefore, the matching algorithm establishes a mapping from definitions in the base model to references in the pointcut graph.

For AoUCM, the matching algorithm does not need to operate on definitions as base model maps and pointcut maps express everything required for the matching process. Therefore, the matching algorithm establishes a mapping from a path node in the base model to a path node in the pointcut map.

A *match* is a set of mappings that, if successful, matches a complete pointcut expression against the base model. In a successful match, all of its mappings contain only unique URNmodelElements, i.e., a URNmodelElement may take part in only one mapping of a successful match. The exception to the rule is the anything pointcut element, which may be matched against one or many base elements. The anything pointcut element may also be matched against zero base elements, i.e., the anything pointcut element may be matched against any sequence of AoUCM base elements including an empty sequence. In this case, however, no mapping exists for the anything pointcut element in the match. Finally, the pointcut expression of one concern may be matched several times in the base model. Therefore, the matching algorithm returns a set of matches as its result for each concern – one match for each matched instance in the base model of the pattern described by the pointcut expression of the concern.

7.1.1 Syntax-Based Matching

The syntax-based matching algorithm compares the static structure of a pointcut expression with the base model. For AoGRL, all intentional elements, actors, and element links are relevant for the matching algorithm. For AoUCM, all path nodes and components are relevant for the matching algorithm.

The matching algorithm is a recursive algorithm that begins at a random start point or reference to an intentional element, actor, or element link of the pointcut expres-

sion. At each step, the algorithm scans all neighboring model elements of the current element of the pointcut expression. Neighboring model elements are considered regardless of the direction of the link or node connection that connects the neighboring model element with the current element. If the matching algorithm finds a matching element in the base model for the initial model element of the pointcut expression, the matching algorithm scans the base model and the pointcut expression in parallel. At each step, the algorithm tries to match all neighboring elements of the pointcut expression against all neighboring elements in the base model. This match requires all possible permutations to be considered. For example, let us assume that the matching algorithm has mapped an OR-fork from the pointcut map to an OR-fork in the UCM model and both OR-forks have two branches. Then, the first branch of the OR-fork on the pointcut map may be matched either against the first or second branch of the OR-fork in the UCM model. The same applies to the second branch. If more than one permutation can be matched, the matching algorithm will continue to explore recursively each matched permutation as an individual match candidate. Finally, the matching algorithm takes cycles in pointcut expressions into account to avoid infinite loops by not further considering the neighbors of already visited pointcut elements.

At each step, new mappings are added to the match result if the matching is successful. If the matching is not successful, the mappings established for the current match candidate are discarded and the match candidate is not pursued further. The matching, however, continues with all other still valid permutations and their corresponding mappings. Matching is not successful if:

- The set of neighboring model elements of the pointcut expression cannot be matched against the set of neighboring model elements in the base model. The number of elements in the set of pointcut elements must be equal or smaller than the number of elements in the set of base elements. For example, this allows matching an OR-fork with two branches on the pointcut map against an OR-fork with two or more branches in the base model as long as two of these branches can be matched.
- A new mapping contradicts the already established mappings for the match candidate. This contradiction may happen because all model elements except for the anything

pointcut element may be mapped only to one other model element in a successful match.

Figure 112 shows an example of the mappings of two successful matches established by the matching algorithm for an AoGRL (right) and an AoUCM model (left). For AoUCM, the responsibilities are mapped to each other since the wildcard matches any name (mapping 2). Mappings 1 and 3 contain unnamed start and end points. In terms of the matching algorithm, one can think of unnamed start and end points as free matches. They may be matched with anything as long as they are matched to the path nodes that are closest to the other mappings. For AoGRL, the dependency links are matched (mapping 2) as well as the goals that are the source and target nodes of the dependency because the names with the wildcard match the names in the base model (mapping 1 and 3).

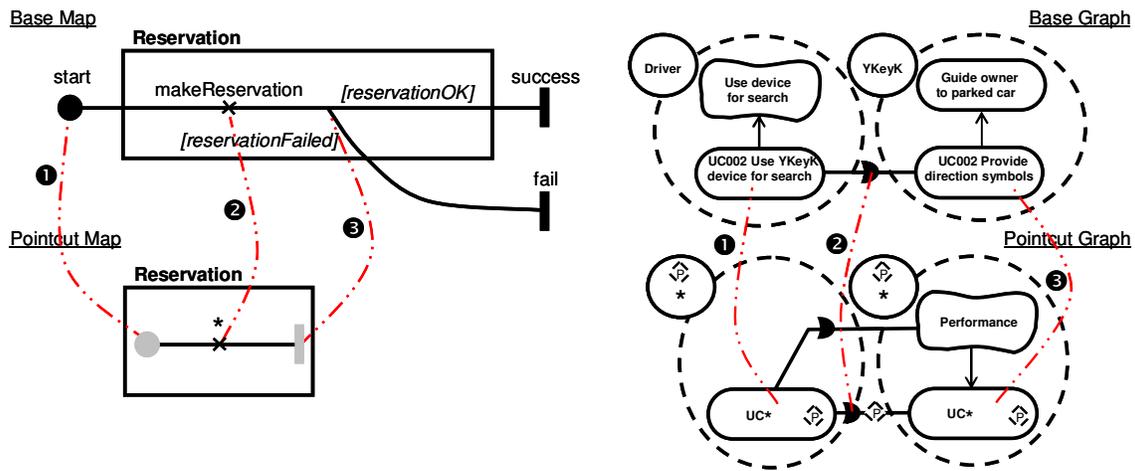


Figure 112 Mappings Between Pointcut Elements and Base Model

As mentioned earlier, contradictory mappings are a reason for unsuccessful matches. This situation is illustrated in Figure 113. After matching the start point in the pointcut map to s1 in the base map (mapping 1) and the two OR-forks (mapping 2), the next step attempts to match the neighbors of the OR-fork in the pointcut map with the neighbors of the OR-fork in the base map. In both cases, the neighbors are OR-joins (note that the OR-fork in the pointcut map also has two neighbors – one for each branch as the direction arrow is skipped, but the two neighbors happen to be the same OR-join). For each neighbor individually, a match is possible (mappings 3a and 3b), but these two mappings contradict each other since the same path node in the pointcut map cannot be mapped to two different path nodes in the base map (or vice versa).

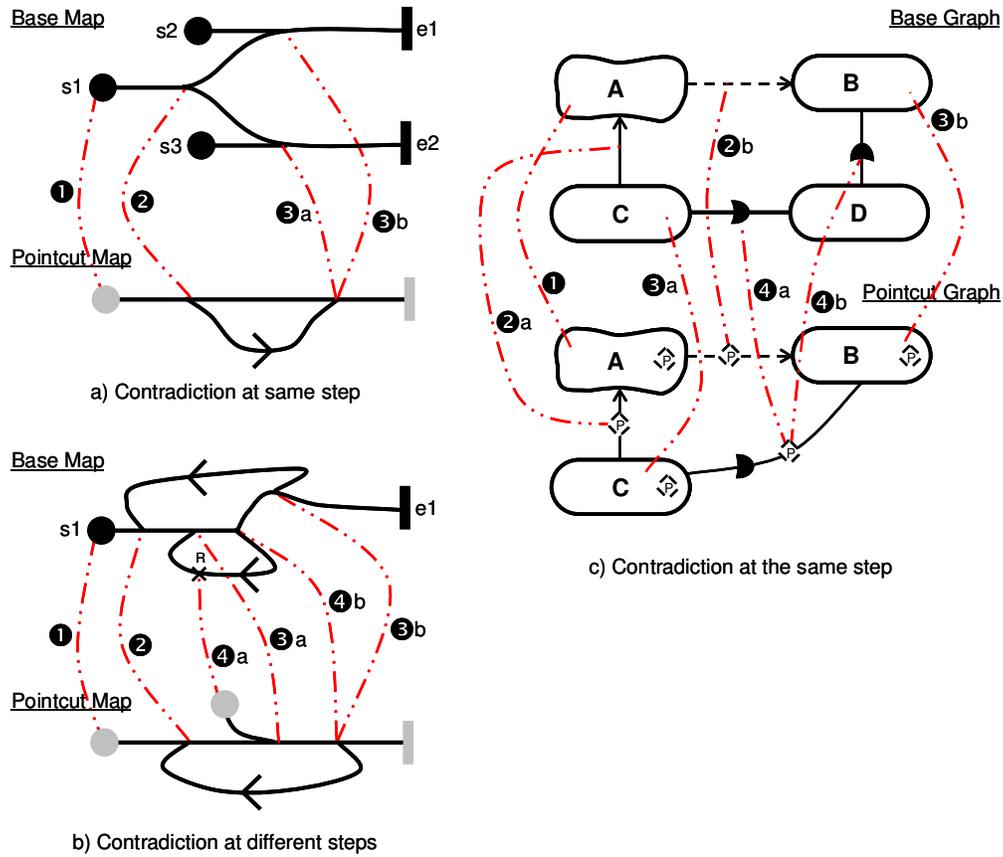


Figure 113 Contradictory Mappings

Figure 113.a shows a contradiction occurring at the same step. Figure 113.b, on the other hand, shows a similar contradiction that appears at different steps of the matching algorithm. After matching the two start points (mapping 1), the two OR-forks (mapping 2), and the four OR-joins (mappings 3a and 3b), the matching algorithm attempts to match the neighbors of the second OR-join in the pointcut map. The start point in the pointcut map can be matched against the responsibility in the base model (mapping 4a) and the second OR-join in the pointcut map can be matched against the third OR-join in the base model (mapping 4b). However, this new mapping 4b contradicts mapping 3b already established by the previous step of the matching algorithm.

Figure 113.c shows a contradiction that appears at the same step of the matching algorithm in an AoGRL model. In the first step, softgoals A are matched (mapping 1). In the second step, the contribution to A and the correlation from A are matched (mapping 2a and 2b). In the third step, goals C and goals B are matched (mapping 3a and 3b). In the fourth step, the dependency from goal C and the dependency to goal B are matched

(mapping 4a and 4b). These two, however, contradict each other because the same link in the pointcut expression is mapped onto two different links in the base model.

The following criteria are taken into account to decide whether a single element of the pointcut expression matches a single base element:

- The types of the model elements must match (this applies to intentional elements, actors, element links, and path nodes).
- A static stub in the pointcut expression may be matched against any other type of stub in the base model except for pointcut stubs (this is useful because often the type of stub is not relevant for the pattern to be matched).
- A stub in the pointcut expression that is not static must be matched against the same type of stub in the base model.
- A neighboring element must be reached from an element in the pointcut expression through the same type of link (decomposition, contribution, correlation, or dependency) or node connection (regular, timeout, or trigger/release branch) and in the same direction as the neighboring base element from the base element.
- The anything pointcut element may be matched to any sequence of path nodes and path node connections in the base model including an empty sequence.
- An intentional element tagged with the anytype pointcut element may be matched to any type of intentional element in the base model.
- An element link tagged with the anytype pointcut element may be matched to any type of element link in the base model.
- Unnamed start and end points in the pointcut expression may be matched to any type of path node in the base model. All other criteria do not apply to unnamed start and end points. They are considered free matches.
- An unnamed end point connected to an unnamed start point in the pointcut expression are not matched but ignored (required by interleaved composition rules).
- The names of the model elements must match (this may require the use of conditions for the mapping, if the model elements are responsibilities or components). If the pointcut expression does not specify a name for a UCM model element except responsibilities, components, start points, and end points, the name is deemed to be the wildcard *.

- The names of conditions must match, if the model elements are start points, waiting places, timers, failure points, or OR-forks. If the pointcut expression does not specify conditions, any condition may be matched.
- The GRL actor of the model elements must be compatible (i.e., for elements within an actor to be matched, their actors also need to match).
- The component hierarchy of the model elements must be compatible (i.e., for elements within a component to match, their components also need to match; as components may be arranged in a hierarchy, the immediate components may not need to be the same for a valid match, as long as the component in the pointcut expression can be found in the component hierarchy of the base element).
- A component of ComponentKind Team (see Figure 163 on Page 305) in the pointcut expression may be matched against any other type of component in the base model (this is useful because often the type of component is not relevant for the pattern to be matched).
- A component in the pointcut expression that is not of ComponentKind Team must be matched against the same type of component in the base model.
- The location of the model elements in their components must match (either first, last, or any location in the component).
- The metadata of the pointcut element must be a subset of the metadata of the base element. Note that metadata matching allows for an annotation-based approach to modeling aspectual models.

Component Hierarchies

Components may contain other components which have to be taken into account by the matching algorithm. AoURN's approach is flexible in that it does not require an exact match of components in the pointcut expression and components in the base model. A match is successful as long as the components in the pointcut expression can be found in the component hierarchy of the base model. Figure 114 shows three pointcut expressions with components that match against the base model at the top left, even though the component hierarchies in the pointcut expression and the base model are not an exact match. While the component hierarchy in the base model shows C2 between C1 and C3, the

pointcut expressions omit C2 and only state that C3 is contained in C1, which is still true for the base model. Therefore, the match is successful.

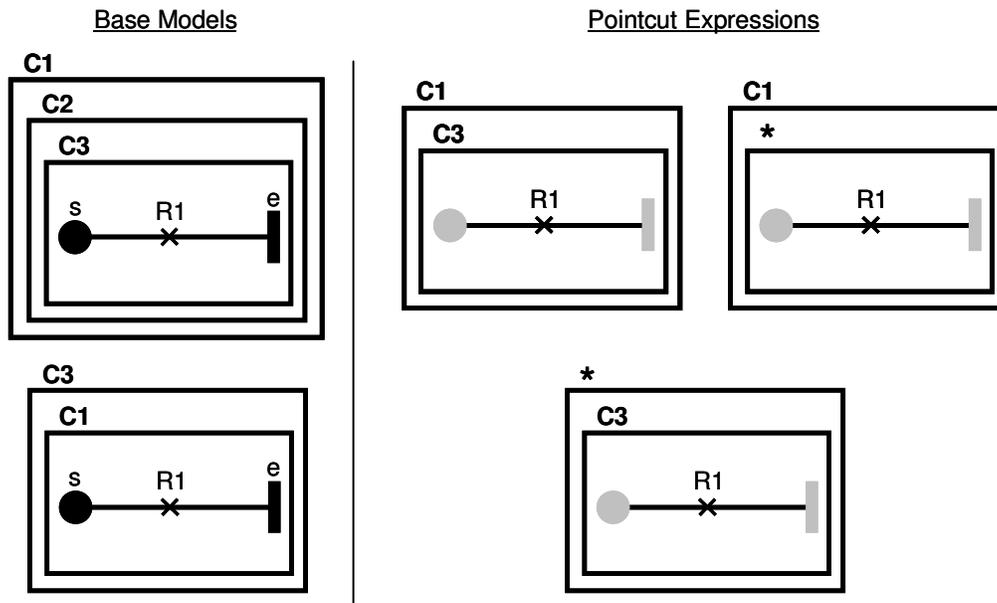


Figure 114 Matching Component Hierarchies

The same three pointcut expressions, however, do not match the base model at the bottom left, because the component hierarchies are not compatible. The first pointcut expression requires C3 to be contained in C1, but the opposite is the case in the base model. Furthermore, the second pointcut expression requires a component to be contained in C1 which is not the case for the base model, and the third pointcut expression requires C3 to be contained in a component which is also not the case for the base model.

Anything Pointcut Element

The anything pointcut element requires special consideration when it comes to a) scanning the base model and the pointcut expression in parallel, b) discarding unsuccessful match candidates, and c) matches that fully contain other matches.

A) Typically, the pointcut expression and the base model are scanned in parallel in a synchronized fashion. When the matching algorithm moves to the next element in the pointcut expression, it also moves to the next element in the base model. A pointcut expression with an anything pointcut element, however, may cause the matching algorithm to remain at the same anything pointcut element while moving on to the next base ele-

ment, because the anything pointcut element may be matched against more than one base element.

B) Normally, an unsuccessful match candidate is simply thrown away. If an anything pointcut element is part of the match, then, instead of discarding the match candidate, the first element after the anything pointcut element must be added to the match for the anything pointcut element and the match continued from that position. Figure 115 illustrates this situation. The example shows the matching algorithm attempting to match the pointcut expression against the base model starting with the first element. For the pointcut expression without the anything pointcut element (Figure 115.a), the first elements can be matched but the second elements do not match. Therefore, the match candidate is thrown out.

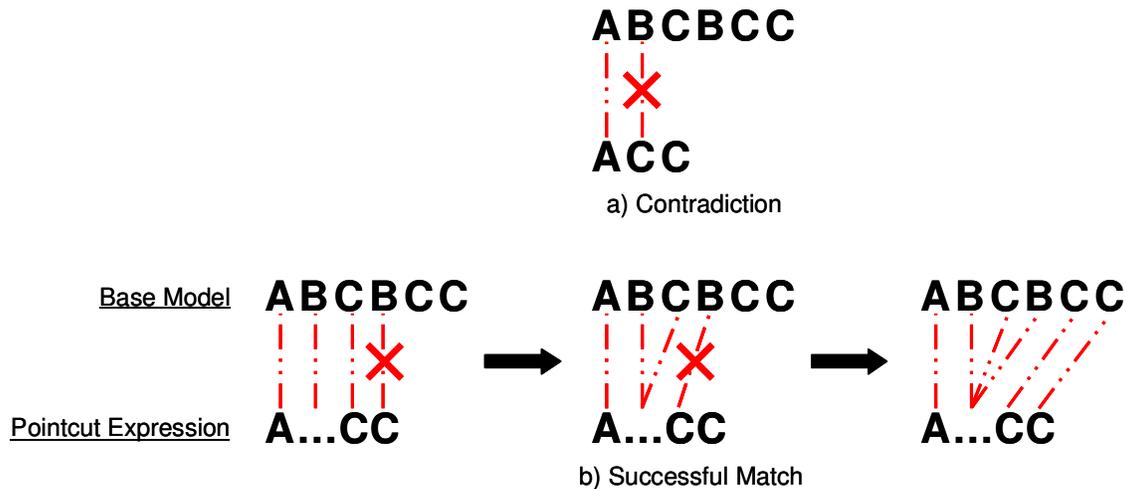


Figure 115 Contradictory Mappings and the Anything Pointcut Element

For the pointcut expression with the anything pointcut element (Figure 115.b), the first three elements can be matched initially but the fourth elements do not match. Instead of throwing out the match candidate, the first element after the anything pointcut element (i.e., the third element) must be added to the mappings of the anything pointcut element. The fourth base element is then re-matched but again fails. It is therefore added to the mappings of the anything pointcut element. Now, the fifth base element is matched against the third pointcut element and also the sixth base element against the fourth pointcut element, completing a successful match of the pointcut expression that includes the anything pointcut element.

C) Finally, pointcut expressions with the anything pointcut element may result in several matches of different lengths. It can happen that a match is fully contained in another match. For example, consider the base model sequence A B C A C C and the pointcut expression A ... C C. The pointcut expression matches the whole sequence as well as the last three elements of the base model by themselves. Since the second match is fully contained in the first match, the first, longer match is discarded to avoid an explosion of matches and duplicate application of a concern at the same location in the base model. This criterion, however, is not the same as keeping the first match, as a shorter, fully contained match may be found after a longer one. It is also not the same as keeping the shortest match, as longer matches may exist that do not fully contain the shortest match.

7.1.2 Enhanced Matching Based on Semantics

One problem faced by AOM is the fragile pointcut problem [29][67][74] – the patterns that describe where in the base model an aspect is applied are often very susceptible to rather small changes in the base model. A small change may be enough for the pattern to no longer match and the aspect not being applied as desired. AoURN's matching and composition mechanisms address this problem for a specific set of changes, i.e., refactoring operations which do not change the meaning of the model but only its syntactic representation [92]. A requirements engineer should rightly expect that such operations do not affect the specification and impact of an aspect.

As a refactoring operation transforms one model into a semantically equivalent model, the semantics-based matching algorithm takes into account semantic equivalences that exist in the URN models. A prerequisite of this approach is a clear semantic definition of URN (Chapter 3).

Several types of semantic equivalences exist in URN models. Type I, the most straightforward equivalence type, involves *whitespace* such as direction arrows (>), empty points (○), and connected end and start points (●) in UCM models (Figure 116.a). These elements are semantically insignificant because they exist for visualization purposes only. They can simply be ignored by the matching algorithm. Inserting or removing any of these elements into or from a URN model, will not affect AoURN's matching and composition mechanism.

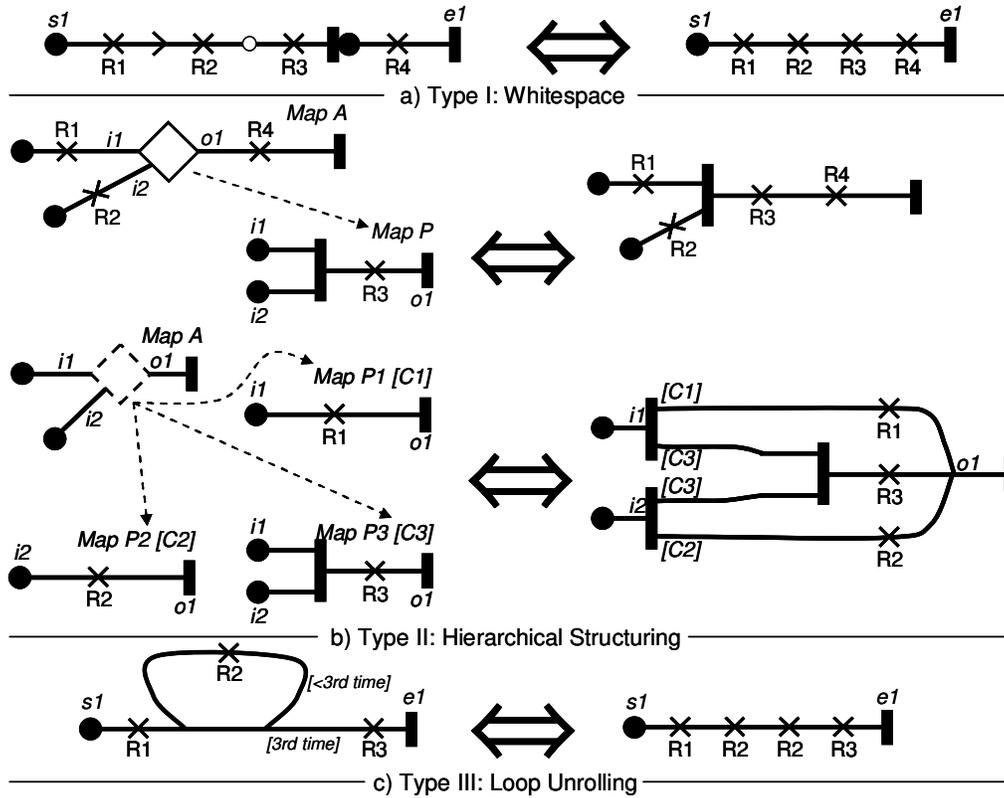


Figure 116 Types of Semantic Equivalences in UCM Models

Type II equivalences involve hierarchical structuring with static, dynamic, or synchronizing stubs in UCM models. Flattened models that are equivalent to all three types of stubs are shown in Figure 14 and Figure 116.b. The GRL example in Figure 117 shows a chain of decompositions. It should be possible to match a longer chain (the one on the left) in the base model against a shorter chain (the one on the right) in the pointcut expression. Note that a stack of intentional elements is shown to indicate that any intentional element may be part of the decomposition chain. While a GRL decomposition chain is neither a true equivalence nor a true refactoring operation, because a shorter chain in the base model must not be matched against a longer chain in the pointcut expression, it is nevertheless discussed here as it relates to hierarchical structuring. Allowing decomposition chains to be matched flexibly is akin to the anything pointcut element in AoUCM. The GRL adaptation of it, however, is an implicit and much more restricted version. The anything pointcut element allows any sequence of path nodes to be matched, if the anything pointcut element is used explicitly in the pointcut expression. Matching of decomposition chains, on the other hand, implicitly allows a series of decompositions and intentional

elements in the base model to be matched against a single decomposition in the pointcut expression.

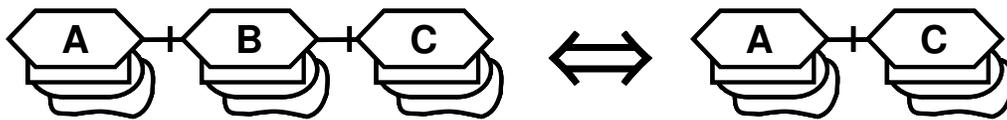


Figure 117 GRL Decomposition Chains

In summary, the particular refactoring operations that are supported by AoURN for type II equivalences are extracting a plug-in map and inlining a plug-in map (the reverse of the first). These types of operations are applicable to most modeling notations as some form of hierarchical structuring is usually provided that can benefit from extracting and inlining. With respect to the discussed semantic equivalences of type I and type II, AoURN is hence a refactoring-safe, aspect-oriented modeling environment.

Type III equivalences in UCM models (Figure 116.c) cover loop unrolling which is discussed in detail by Klein *et al.* [72], who weave UML sequence diagrams by matching semantically equivalent but syntactically different sequences. The authors give a thorough explanation on how to deal with loops but do not address equivalences related to hierarchical structuring. While the findings of Klein *et al.* could be incorporated into the AoUCM approach, AoUCM focuses on hierarchical structuring because it is a much more common refactoring operation in AoUCM models than loop unrolling based on a decade of experience in creating and maintaining UCM and AoUCM models. In other work in the area of semantic-based aspect weaving, Chitchyan *et al.* [37] use natural language processing to take into account English semantics when composing textual requirements documents. A natural language-based approach could be combined with AoURN's matching approach as names of model elements must be matched. Cottenier *et al.* [40][163] match state machines at different levels of abstraction by performing a static control flow analysis to find matching patterns. Patterns are expressed with state machines at the behavioral specification level and then matched against more complex implementation state machines while taking the semantics of the behavioral specification into account. This orthogonal control flow-based approach could also be combined with AoURN's matching algorithm for the matching of UCM models. A recent taxonomy of syntactic and semantic matching mechanisms for AOM models [99] identifies various

pattern matching techniques in the AOM context and highlights the potential influence of approximation-based techniques from the database research community on more sophisticated matching mechanisms.

A first intuition is to use only the flattened model as the basis for the semantics-based matching algorithm, thereby reducing each UCM model to its normalized form. In this case, pointcut expressions cannot match against stubs since flattened models do not contain stubs. However, there is no good reason to exclude stubs from pointcut expressions since a requirements engineer may want to match stubs explicitly. Therefore, the semantics-based matching algorithm distinguishes between model elements that can always be ignored (i.e., direction arrows, empty points, connected end and start points) and model elements that potentially can be ignored depending on the context (i.e., stubs and the start and end points on their plug-in maps as well as decomposition chains).

Figure 118 shows the effects of semantics-based matching of a pointcut map against the base model. Consider the result of the matching algorithm for pointcut map P_a when flattening base model B_i into a single map consisting of a start point $S1$, the four responsibilities $R1$, $R2$, $R3$, and $R4$ in a row, and an end point $E1$. The flattened UCM model is matched three times by the pointcut map ($S1-R1-R2-R3$; $R1-R2-R3-R4$; $R2-R3-R4-E1$). Note that the first and last mappings of each match are with the grey start and end points of the pointcut map.

When applied to the non-flattened base model B_i , the matching algorithm must again consider each base element as a potential start of a match candidate. Therefore, the first match candidate of the non-flattened parent map starts at $S1$ and is followed by $R1$. The next two path nodes, the stub $ST1$ and the start point $S2$, are ignored due to type II equivalences, allowing $R2$ and $R3$ to complete the first successful match of the pointcut map P_a (i.e., $S1-R1-R2-R3$).

The second match candidate starts at $R1$ and then must immediately ignore two elements due to a type II equivalence – $ST1$ and its start point $S2$ in this case. However, this match candidate is equivalent to the match candidate starting at $S2$. To avoid duplicate matches, the match candidate starting at $R1$ is not further considered.

The remaining match candidates for the parent map starting at $ST1$, $R4$, and $E1$ do not return a match. The matching algorithm then examines the plug-in map and finds a

successful match starting at S2 and ending at E2 (S2-R2-R3-E2). Another successful match starting at R2 (R2-R3-R4-E1) is found because the end point E2 is also ignored due to a type II equivalence. The remaining match candidates starting at R3 and E2 are not further considered for the same reason as the match candidate starting at R1 in the earlier example for base model B_i. Consequently, the matching algorithm found equivalent matches for the flattened and non-flattened model.

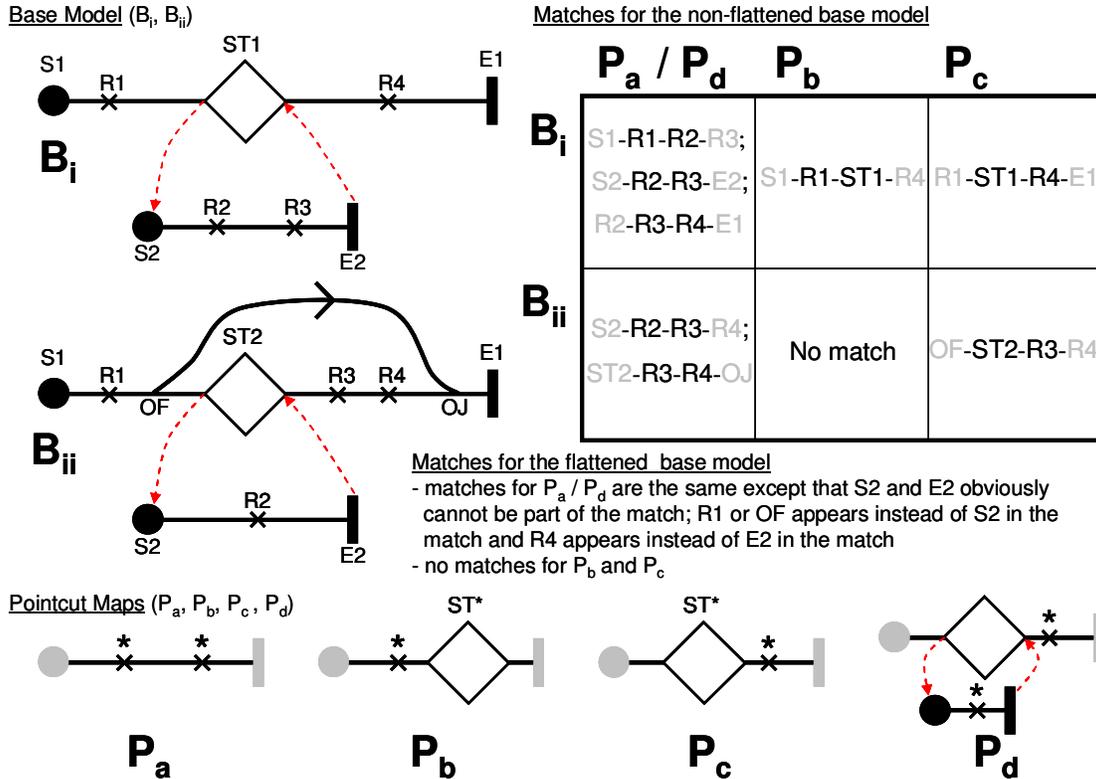


Figure 118 Semantics-Based Matching of Static Stubs

An attempt to match pointcut map P_a against base model B_{ii} starting at R1, OF, ST2, R3, R4, OJ, and E1 yields (ST2-R3-R4-OJ) as the first successful match. A matching attempt of the plug-in map starting at S2, R2, and E2 is also successful and results in (S2-R2-R3-R4). The match candidates starting at OF and R2 are not further considered for the same reason as the match candidate starting at R1 in the earlier example for base model B_i.

Finally, if the pointcut map contains a static stub as shown in the two examples P_b and P_c, then the static stubs in the base models must not be ignored. Therefore, the matches against the base models are (S1-R1-ST1-R4) for pointcut map P_b as well as (R1-ST1-R4-E1) and (OF-ST2-R3-R4) for pointcut map P_c. However, if the requirements en-

gineer adds a plug-in map to the stub in a pointcut map as shown in P_d , the normal form for the pointcut map is used by the matching algorithm. P_d is therefore equivalent to P_a as it is deemed that the requirements engineer uses the stub to structure the pointcut expression hierarchically and not to explicitly match a stub.

Semantic Equivalences of UCM Aspect Markers

The flattened equivalents of static, dynamic, and synchronizing stubs as shown in Figure 14 and Figure 116.b are used by the matching algorithm for an alternative match. For the purposes of the matching algorithm, static AoUCM aspect markers are similar to regular static stubs, because the flattening of static aspect markers occurs in the same way as for regular static stubs. Dynamic AoUCM aspect markers, on the other hand, have a flattened model that is different than the one of other types of stubs. A dynamic aspect marker is required if several aspects insert their aspectual properties at the same location and no precedence rules are defined among them in the concern interaction graph. In this case, the ordering of the aspects is considered random but with only one aspect executing at a time, as is defined by the flattened model for a dynamic aspect marker with the help of a protected component. Since a protected component enforces the one-active-path-at-a-time rule, only one of the aspects may be active at any time. The AND-fork, on the other hand, ensures that the order of the aspects is random while the AND-join ensures that the base behavior may occur only once all aspects have finished executing. Given the precedence rules specified as A before B before C and D in any order before E in the concern interaction graph, and a composition rule for the aspects that adds aspectual behavior before R2, Figure 119 shows the base model with three static aspect markers for A, B, and E and one dynamic aspect marker for C and D. The equivalent flattened model is presented at the bottom right. Semantics-based matching uses the flattened model to match aspect markers and hence is more independent from the aspect-oriented structuring of the model.

While other interpretations of a dynamic aspect marker are possible (e.g., a true parallel execution of all aspects or an error may even be issued), the one presented here is chosen because similar interpretations exist for other AOM and AOP approaches. For example, AspectJ's interpretation is that the order in which aspects are executed is unde-

fined and hence the aspects are executed in random order but not at the same time when precedence rules are not given explicitly.

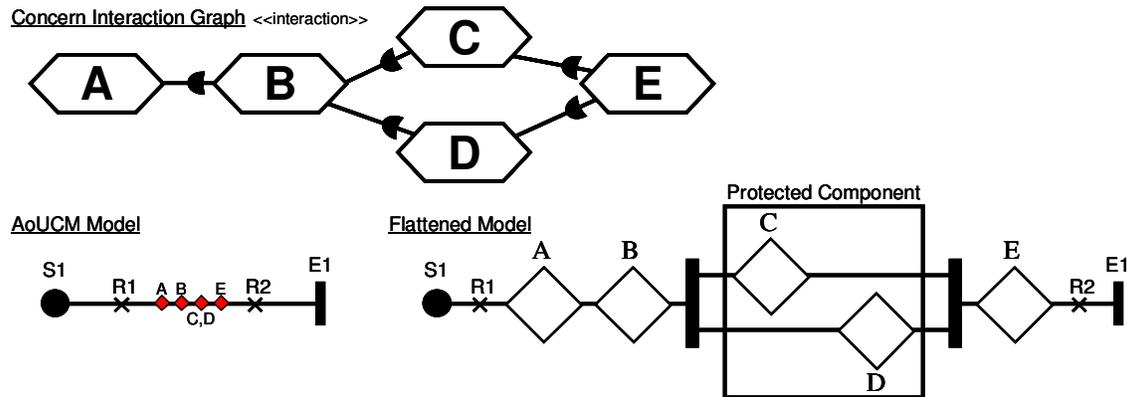


Figure 119 Flattened Dynamic Aspect Marker

Other equivalences may be added to the matching algorithm by first defining the equivalence as done for static stubs or aspect markers, and then updating the way the affected path nodes are handled in the matching algorithm. Candidates for new equivalences are OR-joins, which possibly could be ignored, and waiting places with conditions that could possibly be interpreted as start points with conditions for matching purposes. At this point, however, AoURN does not support these candidates.

Updated Matching Criteria for Semantics-Based Matching

Using flattened models enables matching at a more semantic level as an exact syntactic match is not required. For example, a dynamic stub in the base model is now not only matched by a dynamic stub in the pointcut expression but also by a pointcut expression with an AND-fork and an OR-join. Therefore, the initial matching criterion that the type of path nodes must be the same is relaxed by the following criteria:

- Direction arrows, empty points, and connected end and start points are ignored in the pointcut expression and in the base model.
- A static stub and the start and end points of its plug-in map may be ignored in the base model.
- An AND-fork in the pointcut expression may be matched against a dynamic or synchronizing stub and the start points of its plug-in maps in the base model (when entering the stub). Note that the start points are actually ignored in this case.

- An OR-join in the pointcut expression may be matched against many end points with a plug-in binding for the same dynamic stub in the base model (when exiting the stub).
- An AND-join in the pointcut expression may be matched against many end points with a plug-in binding for the same synchronizing stub in the base model (when exiting the stub).
- A GRL decomposition link in the pointcut expression may be matched against a GRL decomposition link followed by zero, one, or more pairs of intentional elements and decomposition links.

The impact on the matching algorithm to be able to address the above new criteria is rather small. Some elements are simply ignored. With respect to the matching algorithm, it is as if these elements do not even exist. For other elements, the new criteria describe alternatives for a match at a particular path node. Instead of checking for one possible match, the algorithm now simply checks for another one if the syntactic match is unsuccessful. Furthermore, the matching algorithm may now match an AND-fork in the pointcut expression against a dynamic or synchronizing stub and then ignore its start points, thus applying a combination of the above two mechanisms. Finally, decomposition chains are similar to the matching of anything pointcut elements. At each matching step for the anything pointcut element, any path node may be added to the list of elements matched against the anything pointcut element. In the case of a decomposition link, an intentional element may be added to the list of matched elements for the decomposition link, if a decomposition link was added last. If an intentional element was added last, then another decomposition link may be added to the matched elements. All of these adaptations, however, do not change the fundamental structure of the matching algorithm.

7.2. Composition Algorithm

Requirements engineers need to be able to reason about the composed system. The composed model hence needs to be automatically constructed to help the modeler understand the overall behaviour and assess the impact of aspects on the model. This involves first constructing the composed model but also then visualizing the composed model. Finding

the right layout for the composed model is a difficult problem for most aspect-oriented modeling notations as the layout must be intuitive to the modeler. The AoGRL notation to a certain extent and certainly the AoUCM notation with its paths bound to components have proven to be difficult to lay out automatically. This layout problem equally applies to the GRL and UCM notations. jUCMNav's auto-layout mechanism works only for rather simple models. An effective composition mechanism therefore manages to correctly compose concerns and then adequately visualize the composed model. AoURN's composition algorithm addresses these two points.

The composition algorithm makes use of the matches provided by the matching algorithm to apply concerns to the base model. While the matching algorithm is basically the same for AoGRL and AoUCM, the composition algorithms are quite different for these two notations, because aspect markers are used very differently. What is similar, however, is that aspect markers are added to the base model to indicate insertion points for aspectual properties, as is described for AoGRL in Section 7.2.1 and for AoUCM in Sections 7.2.2 to 7.2.9. Aspect markers enable the navigation of AoURN models in an aspect-oriented way with AoViews as introduced in Section 6.4. The AoURN model may also be composed into a traditional URN model as illustrated in Section 7.2.11, while conflicting aspect are discussed in Section 7.2.10.

While the details differ, the general approach to composition is the same for AoGRL and AoUCM. First, the condition of a concern must evaluate to true for the concern to be applied. Second, the precedence rules defined by the concern interaction graph introduced in Section 6.1.5 must be obeyed to avoid conflicts and dependency problems between concerns. Therefore, composition of concerns unfolds in several *waves* based on the order in which concerns have to be applied to the URN model as defined by the concern interaction graph (Figure 120).

In each wave, all concerns that are not restricted by precedence rules are applied to the URN model (i.e., all nodes in the concern interaction graph that are not the source of any dependency or correlation link). At the end of each wave, the concerns that were applied are considered removed from the concern interaction graph, leading to new concerns without outgoing links. These concerns will then be applied in the following wave. To ensure that all concerns of a wave are applied to a consistent base model, the match-

ing algorithm first establishes the mappings for all concerns of a wave before the composition algorithm changes the model one concern at a time. Note that any concern that is not shown on the concern interaction graph is part of the first wave.

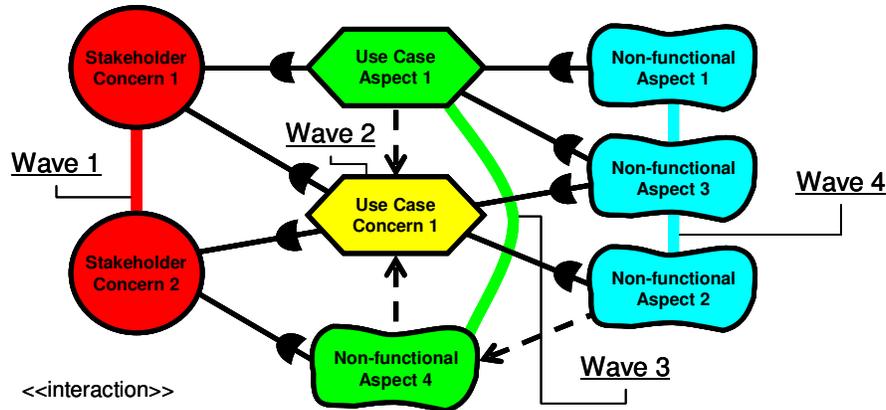


Figure 120 Order of Concern Composition

For example during the third wave in Figure 120, the Use Case Aspect 1 and the Non-functional Aspect 4 are applied to the AoURN model that includes the aspect markers and aspectual properties of all concerns applied during the first and second waves (i.e., those concerns not shown on the concern interaction graph, the Stakeholder Concern 1, the Stakeholder Concern 2, and the Use Case Concern 1). It also includes the aspectual properties of Use Case Aspect 1 when Non-functional Aspect 4 is applied and the aspectual properties of Non-functional Aspect 4 when Use Case Aspect 1 is applied, but it does not include their aspect markers (i.e., the result of their composition).

If cyclical relationships exist among concerns, no concern without outgoing links may be found in the concern interaction graph even though some concerns may not have been applied yet. In this case, composition terminates with an error. Once all concerns are applied to the model, the transient results of the matching algorithm may be discarded.

As is typical for AOM techniques – and AoURN is no exception – being applied first or having higher priority does not mean that the concern executes first in the composed model as illustrated in Figure 121. The execution order depends on whether the concern is applied before or after a join point. The concerns Authentication (A), Logging (L), and Transaction Support (T) all have to be applied before and after a join point (e.g., performService). Smart card-based authentication needs to check a PIN before and eject the smart card after, logs need to be saved before and after, and transactions need to be

initiated before and closed after. Given the conflicts among the three concerns defined in the concern interaction graph in Figure 121, the order of execution is A, L, and then T before the join point and T, L, and then A after the join point after the third wave of compositions. The final composed behavior occurs as follows. Authentication is checked and if it fails, the scenario continues with the second aspect marker A, thus skipping all other features. If authentication succeeds, the start of the service is logged, and then a transaction is initiated. Note that logging should not be part of the transaction, because logging must not be undone in case the service fails. The service then occurs and transaction support ensures proper clean up if the service fails. If the service does not fail, the transaction is closed. In any case, the end of the service is logged, and the smart card is ejected.

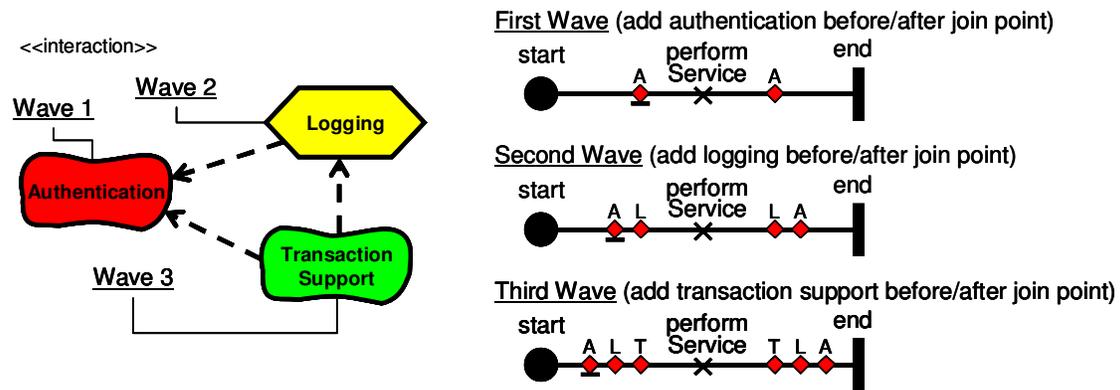


Figure 121 Order of Concern Execution

7.2.1 Insertion of AoGRL Aspect Markers

For AoGRL, the majority of the work required for composition is already done by the matching algorithm. The results of the matching algorithm are several matches, each containing one or more mappings. AoGRL composition unfolds in two simple steps. First, determine the intentional elements and actors to which an aspect marker needs to be added and create the metadata for it. Second, encode the mappings from the match in metadata, so that this information is available for off-line visualization. In the context of the AoURN composition algorithms, off-line means that the matches and mappings established by the matching algorithm are not available anymore. Essentially, the composition mechanism translates the transient data structures used by the matching algorithm into a format that can be saved in an AoURN model.

The candidates for aspect markers are all base elements with a mapping in the match. If such a base element is mapped to a pointcut element that is connected to a non-pointcut element or to a pointcut element tagged with the pointcut deletion marker (i.e., the base element is transformed by the aspect), then the base element is tagged with an aspect marker by adding the metadata name/value pair *aspect/aspect marker* to the definition of the base element. For example, given a match with the three mappings from Stakeholder Softgoal A1 to *, from Stakeholder Softgoal B to * Softgoal B, and from Stakeholder Goal C1 to Stakeholder Goal C1 in Figure 122, an aspect marker is added to the softgoal Stakeholder Softgoal A1 and the goal Stakeholder Goal C1 but not to softgoal Stakeholder Softgoal B. The aspect markers are added because the softgoal * is connected to the non-pointcut element Pointcut Specific Softgoal and goal Stakeholder Goal C1 is connected to the non-pointcut element Task of Aspect. Both are also connected to the correlation with the pointcut deletion marker. No aspect marker is added to the softgoal Stakeholder Softgoal B, because * Softgoal B is connected only to the OR decomposition which is part of the pointcut expression. No aspect marker is added to the softgoal Stakeholder Softgoal B, because * Softgoal B is connected only to the OR decomposition which is part of the pointcut expression.

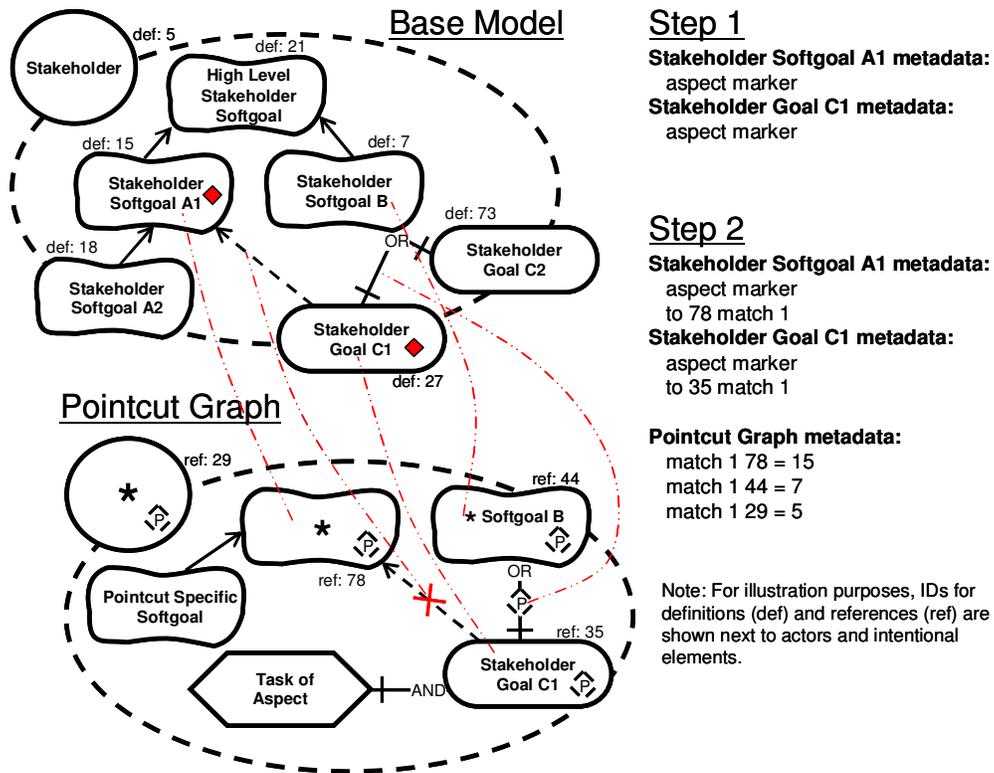


Figure 122 Composing AoGRL Concerns

Note that the definition of a base element is tagged and not a particular reference to the definition. Tagging the definition allows aspect markers for an intentional element or actor to be shown for any reference to the intentional element or actor in the base model, respectively, as it is important for the requirements engineer to know that an element is affected by a concern wherever it is used.

The first step is sufficient for the off-line visualization of aspect markers, but does not provide enough information for the creation of off-line AoViews. The second step hence encodes all required mappings to visualize AoViews. The required mappings are the mappings of any base element with an aspect marker and the mappings of any parameterized pointcut element.

The first kind of mapping defines which aspect actually causes the insertion of the aspect marker. It is realized by the metadata name/value pair *aspect/to id match N* assigned to the base element with the aspect marker. *id* is the unique model identifier of the pointcut element mapped from the base element (the reference is used) and *N* is a unique number for the match. The unique match number identifies a particular match out of the many matches that could exist for the pointcut graph identified by the pointcut element.

The second kind of mapping allows parameterized pointcut elements to be replaced by the actual match in the AoView. It is realized by the metadata name/value pair *aspect/match N id₁ = id₂* assigned to the pointcut graph with *N* again being the unique match number, *id₁* being the unique model identifier for the parameterized pointcut element (the reference is used) and *id₂* being the unique model identifier for the matched base element (the definition is used). By assigning this information to the pointcut graph, it does not have to be repeated for each base element with an aspect marker.

For example, the metadata to 78 match 1 is added to Stakeholder Softgoal A1 in Figure 122. When the aspect marker of Stakeholder Softgoal A1 is selected, an AoURN tool understands that the AoView (Figure 74) of the pointcut graph that contains the pointcut element with id 78 needs to be visualized. The AoURN tool further knows that it is match number 1 that needs to be visualized and proceeds to retrieve the substitution information from the metadata of the pointcut graph. Hence, the Task of Aspect and its AND decomposition are highlighted, because they are non-pointcut elements connected to the pointcut element with id 78. The removed correlation is also highlighted, because,

again, it is connected to the pointcut element with id 78. Furthermore, the softgoal * is substituted with Stakeholder Softgoal A1, * Softgoal B is substituted with Stakeholder Softgoal B, and stakeholder * with Stakeholder as specified in the metadata of the pointcut graph for match 1, i.e., match 1 78 = 15, match 1 44 = 17, and match 1 29 = 5, respectively.

Figure 123 shows the updated summary of all metadata required by AoURN including its AoGRL matching and composition mechanisms. The metadata name/value pair aspect/to id match N may be assigned to an IntentionalElement or Actor, while the metadata name/value pair aspect/match N id₁ = id₂ may be assigned to a GRLGraph that is a pointcut graph.

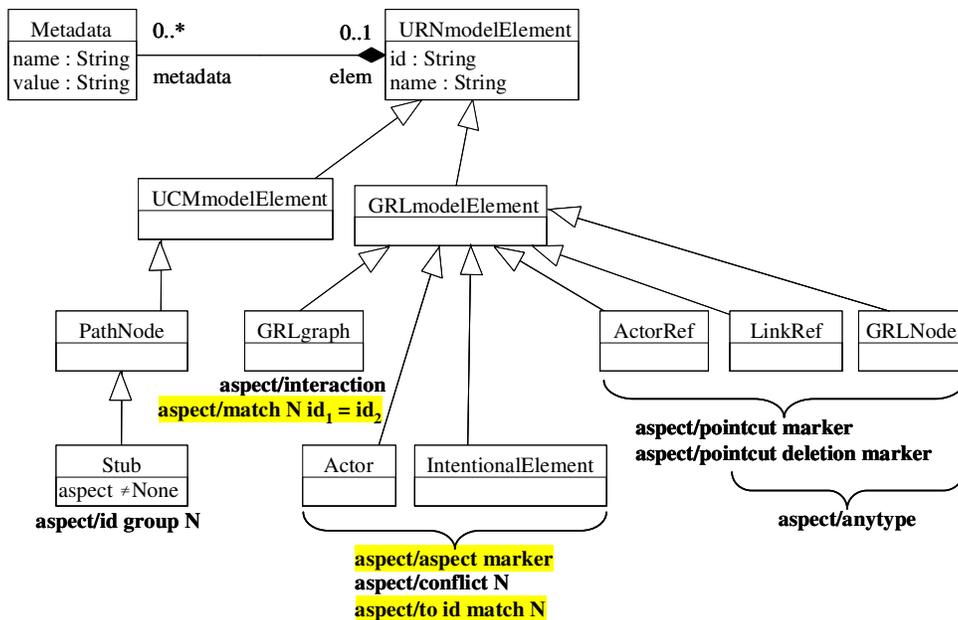


Figure 123 Updated Metadata for AoGRL Composition

There is one special case that needs to be considered for AoGRL composition, as it slightly changes Step 1 of the composition mechanism. If all mapped pointcut elements are connected only to other pointcut elements and pointcut deletion markers are not used in the pointcut expression but there exist non-pointcut elements on the pointcut graph, then aspect markers are added to all mapped base elements to indicate that the aspect adds elements to the GRL model. For example, the non-pointcut element Task of Aspect in Figure 124 is not connected to the pointcut expression, as is no other non-pointcut

element. Therefore, aspect markers are added to all three intentional elements with a mapping to the pointcut graph.

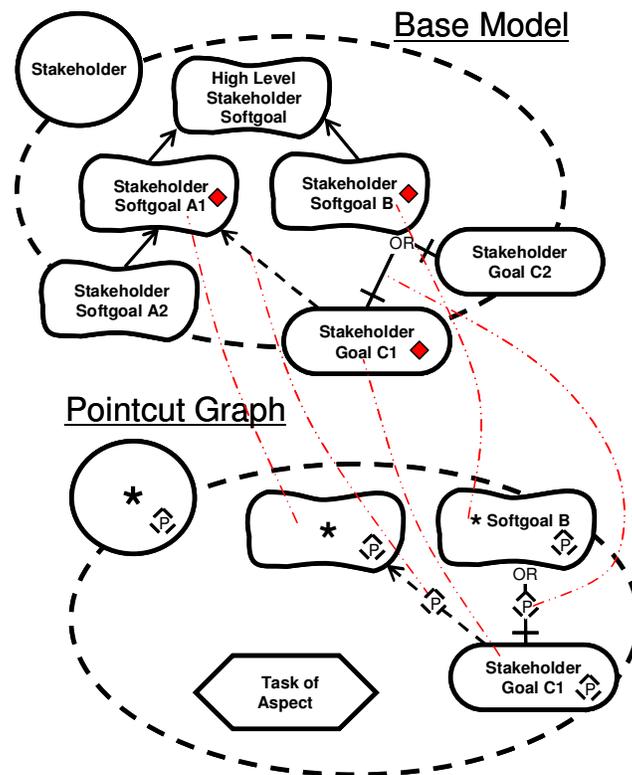


Figure 124 Composing AoGRL Concerns – Special Case

7.2.2 Insertion of AoUCM Aspect Markers – Overview

For AoUCM, composition takes into account several matches, each containing one or more mappings as established by the matching algorithm, and the plug-in bindings of pointcut stubs when adding aspect markers to the base model. Figure 125 gives an example of the 3-step composition mechanism for AoUCM based on the reservation system introduced in Section 6.7.1. The final result of the AoUCM composition algorithm are aspect markers that are added with proper plug-in bindings to the base model as well as additional information encoded in metadata for off-line visualization. For an aspect marker to be added to the base model, the composition mechanism needs to first identify the insertion point where the aspect marker is to be added and second identify the path nodes on the aspect map required for the plug-in bindings with the aspect marker in addition to the in-path or out-path of the pointcut stub. Step 1 of the AoUCM composition

mechanism hence identifies the insertion point and the first path node for the plug-in binding. Step 2 identifies the second path node for the plug-in binding, while Step 3 actually inserts the aspect marker with proper plug-in bindings.

7.2.3 Insertion of AoUCM Aspect Markers – Step 1

Step 1 of the AoUCM composition mechanism identifies the insertion point and the first path node for the plug-in binding. For example, the mappings of a single match established by the matching algorithm and the plug-in bindings of the pointcut stub are shown in Figure 125.a. Step 1 of the composition algorithm creates mappings for the in-paths and out-paths of the pointcut stub, because by definition these are part of the plug-in bindings of an aspect marker. In-paths and out-paths connected to empty path segments containing only whitespace are not considered (e.g., the crossed out path segment in Figure 125.b), because no aspectual behavior is specified by empty path segments and therefore nothing needs to be inserted into the base model. Consequently, aspectual properties are added only after the matched pointcut expression in this example.

The mapping for an in-path is found by following its plug-in binding to the start point on the pointcut map. The mapping of the path node after this start point is retained. The mapping for an out-path is found by following its plug-in binding to the end point on the pointcut map. The mapping of the path node before this end point is retained (see mapping ② in Figure 125.a and mapping ① in Figure 125.b). Note that if a pointcut stub contains multiple pointcut maps, several mappings may be retained for an in-path or out-path. The mapping created for the in-path or out-path of the pointcut stub identifies the join point in the base model to which the aspect is applied as well as the first half of the desired plug-in bindings.

In addition to identifying the join point, the actual insertion point of the join point also needs to be determined (see the large arrow called the *insertion point arrow* in Figure 125.b). The insertion point arrow points towards the path node in the base model identified by the mapping from the start or end point on the pointcut map (depending on which plug-in binding is followed from the pointcut stub), thus indicating which insertion point to use. The insertion point arrow is especially important for base elements that may have more than one path node as successor or predecessor. The insertion point arrow

clearly identifies the successor or predecessor that plays a role in the composition. In terms of the AoURN metamodel, the insertion point arrow is a node connection of the base element.

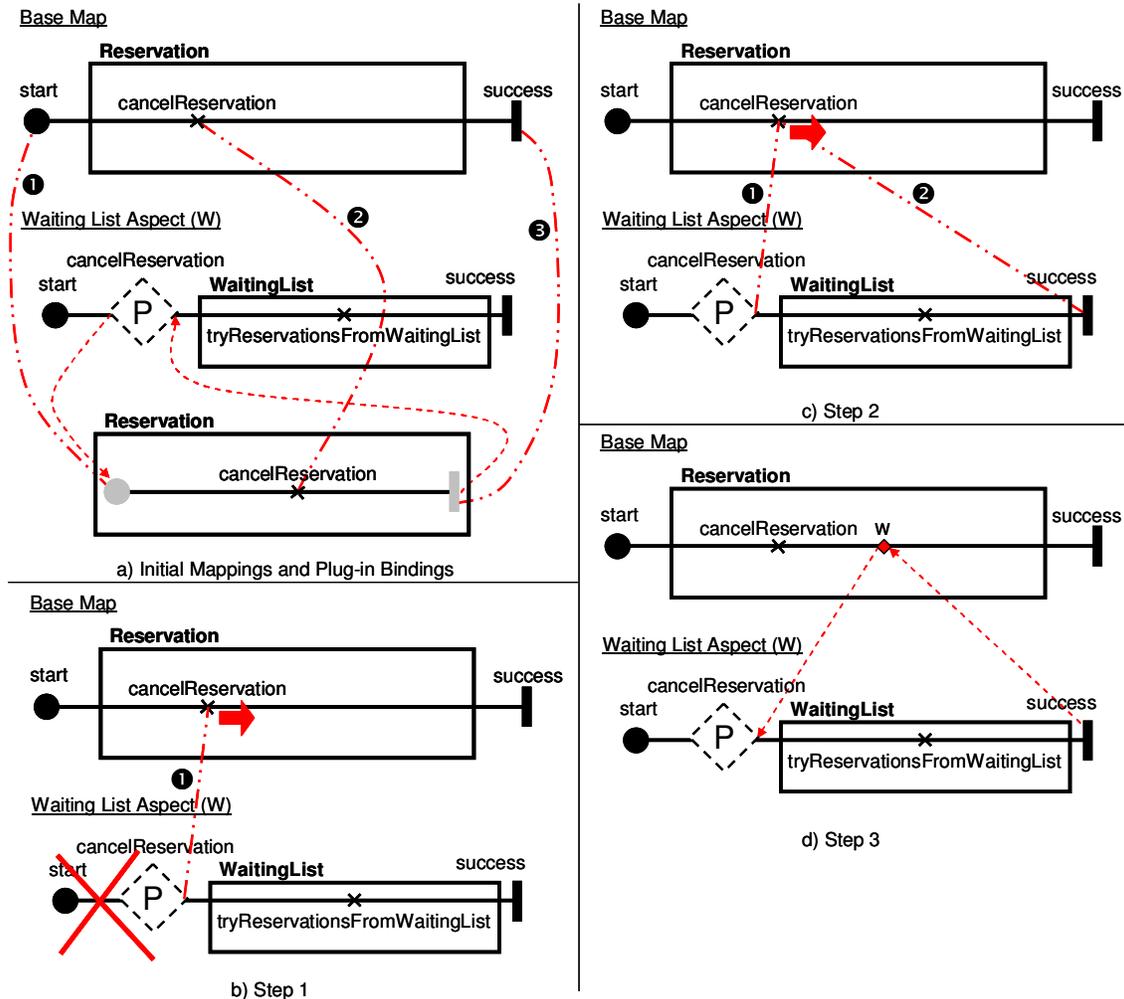


Figure 125 Composing AoUCM Concerns

7.2.4 Insertion of AoUCM Aspect Markers – Step 2

Step 2 scans the path on the aspect map to find a corresponding start point for each mapped in-path of the pointcut stub and a corresponding end point for each mapped out-path of the pointcut stub. The result of the scan leads to the second half of the desired plug-in binding. The path is scanned in the backward direction for in-paths and in the forward direction for out-paths. If a pointcut stub is reached by the scan, the scan does not go beyond the pointcut stub but stops for this branch, because each in-path and out-

path of a pointcut stub is already dealt with by its own scan (see the loop example in Figure 82 on Page 128). Once a start or end point is found, a mapping to the same base model element as for the corresponding in-path or out-path, respectively, is created (mapping ② in Figure 125.c). If not exactly one start or end point can be found, the aspect map is malformed and composition cannot proceed. However, local start points and local end points are not counted by this step. In addition, all start and end points that are not reached by the backward and forward scan, respectively, have to be local start and end points (see Figure 85.c on Page 131 for examples). If they are not, the composition mechanism issues a warning but proceeds with the next composition step.

7.2.5 Insertion of AoUCM Aspect Markers – Step 3

Step 3 inserts the aspect marker in the base map. The insertion point is the one associated with the join point identified in Step 1 and the one towards which the insertion point arrow also identified in Step 1 is pointing. Finally, the mappings created in Step 1 and Step 2 are converted into plug-in bindings for the aspect marker. Several sets of aspect markers may have to be inserted for one aspect (e.g., if the same pointcut expression is matched successfully against many base maps or if one pointcut stub contains multiple pointcut maps that cause multiple successful matches).

7.2.6 Conditional Aspect Markers

Figure 126 introduces a new feature for the reservation system to illustrate the composition of the conditional aspect marker. Instead of using a waiting list, a reservation at a partner hotel is first attempted. The reservation is added to the waiting list only if the reservation at the partner hotel fails. In terms of the composition algorithm, the only difference to the example in Figure 125 occurs at Step 3. A forward scan from the start point used in the plug-in binding of the aspect marker reveals that not all paths lead to the pointcut stub. Therefore, the composition algorithm notices that the pointcut stub may be circumvented on the aspect map, indicating a conditional composition. Because of that, a conditional tunnel aspect marker is inserted before `addToWaitingList` to highlight that the path segment after the conditional aspect marker may be skipped. In the example in Figure 126, this occurs if the reservation at the partner hotel is successful.

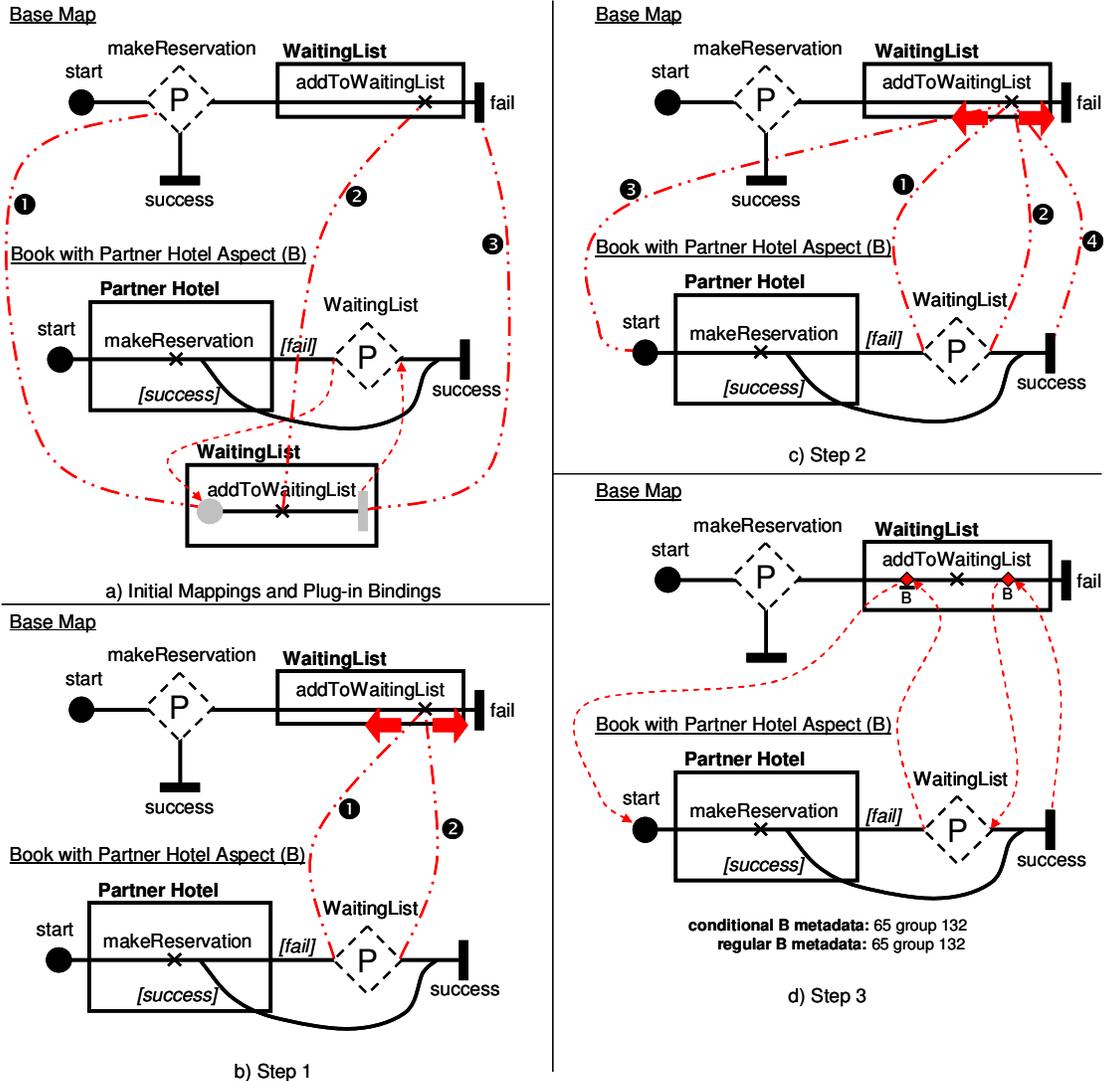


Figure 126 Composing AoUCM Concerns – Conditional

Furthermore, a test needs to be performed to determine whether an aspect marker group is required. This test involves a forward scan from each element on the aspect map that is used in an in-binding of the aspect marker. An aspect marker group is necessary, if the forward scan of element E with an in-binding from aspect marker A reaches at least one element that does not have an out-binding to A but has an out-binding to a different aspect marker than A. In the example in Figure 126, the start point and the out-path of the pointcut stub need to be examined. From the out-path, only the end point can be reached (plug-in binding converted from mapping ④ exists). From the start point, the in-path of the pointcut stub and the end point can be reached (plug-in bindings converted from

mapping ❶ and ❷ exist, respectively). The end point, however, is not used in the plug-in binding of the conditional aspect marker but is used in the plug-in binding of the regular aspect marker. Therefore, an aspect marker group is necessary.

If an aspect marker group is required, the composition algorithm assigns a unique number (e.g., 132) to the aspect marker group and tags all aspect markers (in this case two) with the metadata name/value pair *aspect/65 group 132*, assuming that 65 is the unique model identifier of the aspect map for the Book with Hotel Partner aspect. With the help of this information, the traversal mechanism is able to allow the aspect map to be exited via any aspect marker in the aspect marker group regardless of which aspect marker from the aspect group is used to enter the aspect map.

However, this also means that different aspect markers in the aspect marker group cannot have out-bindings with the same element on the aspect map as this makes the traversal of the aspect map non-deterministic (i.e., which plug-in binding should be chosen to exit the aspect map?). Hence, the composition algorithm checks for this just before Step 3, ensuring that no end point on the aspect map has more than one out-binding, if an aspect marker group is required for the composition. If an end point does have more than one out-binding, then the aspect map is malformed and composition does not occur. Note that in-paths of a pointcut stub cannot have more than one out-binding and therefore do not have to be checked, because the only out-binding created for the in-path is when the in-path itself is examined by the composition algorithm.

7.2.7 Tunnel Aspect Markers (Replacement)

Figure 127 introduces another feature for the reservation system to illustrate the composition of tunnel aspect markers. Tunnel aspect markers are used to indicate replacement in the base model. Let us assume that the hotel experienced a massive failure of its makeReservation service. It is decided to temporarily replace the current service by letting all reservations be handled by the Temporary Forward to Partner Hotel feature which removes the current service and replaces it with its own. Once the crisis is resolved, the aspect can be unplugged from the system and the original reservation service takes control again.

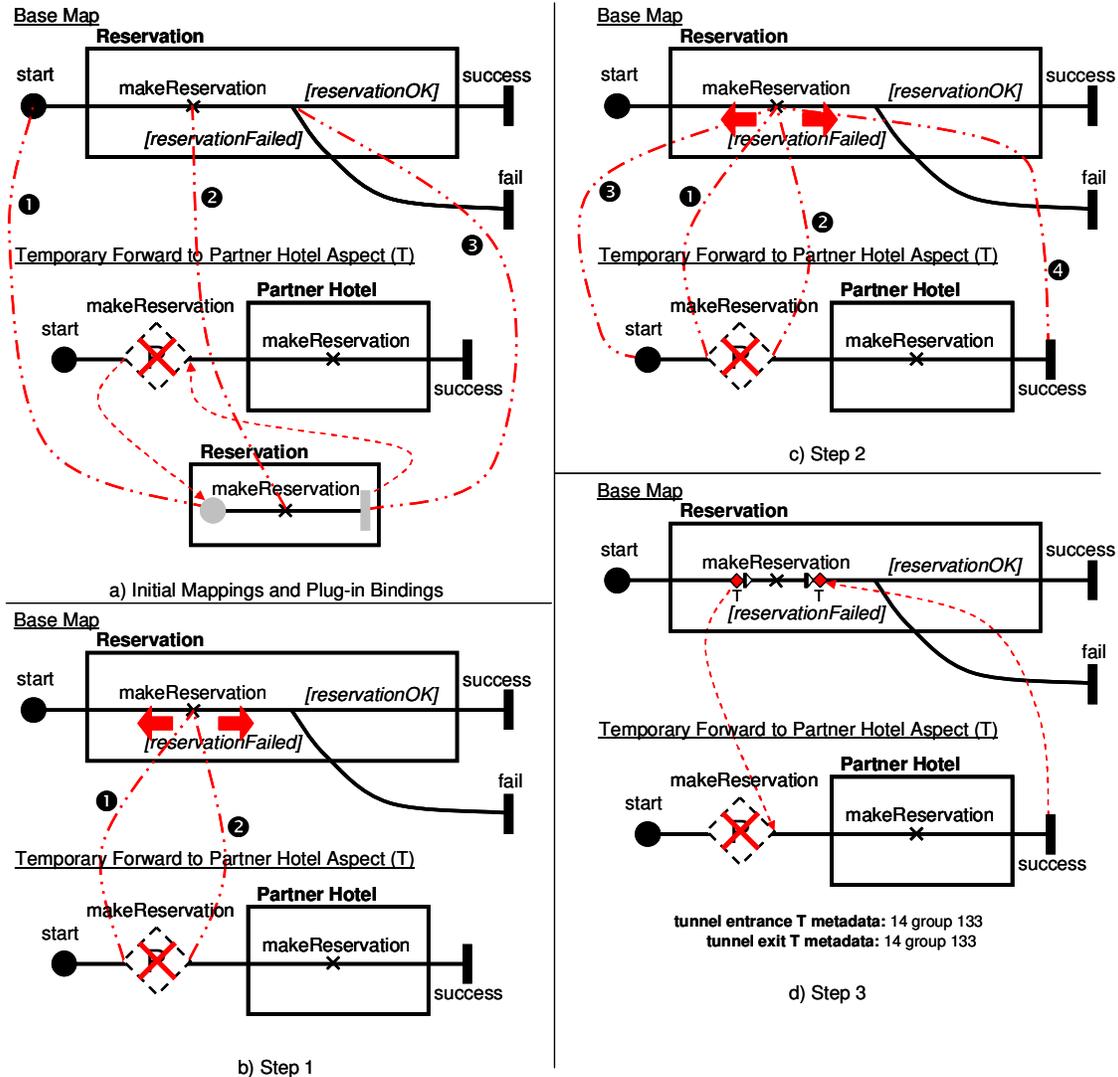


Figure 127 Composing AoUCM Concerns – Replacement

The aspect in Figure 127.a matches against the makeReservation service of the current Reservation system and replaces it with the makeReservation service of the Partner Hotel. The difference for the composition algorithm lies in Step 1 and in Step 3. The composition algorithm realizes that a different mechanism must be used because of the replacement pointcut stub on the aspect map.

In Step 1, the empty segment before the replacement pointcut stub is not discarded. This is done because the composition mechanism needs to establish a mapping to the beginning of the matched pattern, as this is the beginning of the replacement. Note that the pointcut expression in the example contains only one element which leads to the

in-path and out-path of the pointcut stub being mapped to the same base element in Figure 127.b. Pointcut expressions with more elements, however, would see the two mappings point to different base elements, i.e., the beginning and the end of the matched pattern. Also note that an empty path segment after the replacement pointcut stub is also not discarded, thus allowing a removal of base elements without a replacement, i.e., a straight deletion from the model.

While Step 2 is the same, Step 3 is different because tunnel aspect markers have only one plug-in binding instead of two. A tunnel entrance aspect marker is added at the insertion point between mapping 3 and mapping 1 in Figure 127.c and a tunnel exit aspect marker is added at the insertion point between mapping 2 and mapping 4 in Figure 127.c. Mapping 2 is then converted into the in-binding of the tunnel entrance aspect marker and mapping 4 is converted into the out-binding of the tunnel exit aspect marker. Finally, metadata needs to be added to indicate the aspect marker group for the tunnel aspect markers. The example assumes that the unique ID of the aspect map is 14 and the unique match number is 133.

In summary, the type of aspect marker (regular, conditional, tunnel entrance, or tunnel exit) is determined by an examination of the path on which the pointcut stub resides in Step 3 of the composition algorithm. A regular aspect marker is used by default. A conditional aspect marker is used if a choice point on the path causes the pointcut stub to be potentially circumvented. Tunnel entrance and exit aspect markers are used if there is a replacement pointcut stub on the path. Note that there is another situation where tunnel aspect markers could be used. A conditional aspect marker may be upgraded to tunnel aspect markers, if the condition of the choice point leading to the pointcut stub is always false. In the general case, this cannot be determined with a static analysis of the aspect map, but if the condition expression of the choice point is equal to [false], tunnel aspect markers should be used (Figure 71.a on Page 111 illustrates this case).

7.2.8 Pointcut Variables

The example in Figure 127 only replaces the makeReservation service. A massive failure of the reservation system, however, may affect not just this service but all services of the Reservation and WaitingList components. An updated Temporary Forward to Partner Ho-

tel aspect addresses this now by replacing all of those services with its own. It also serves as an example to demonstrate how variables are handled by the composition mechanism. Variables add additional processing to Step 1 and Step 3 of the composition process. While the pointcut expression now matches several more instances of the pattern, the example in Figure 128 still focuses just on one match – the one with makeReservation.

In Step 1, the mapping from any element with a variable to the matched base element is copied over to where the variable is used in the aspect map. For example, the mapping from \$service = * in the pointcut map to makeReservation in the base model is copied to the \$service responsibility in the aspect map, i.e., \$service is now mapped to makeReservation (mapping ② in Figure 128.b).

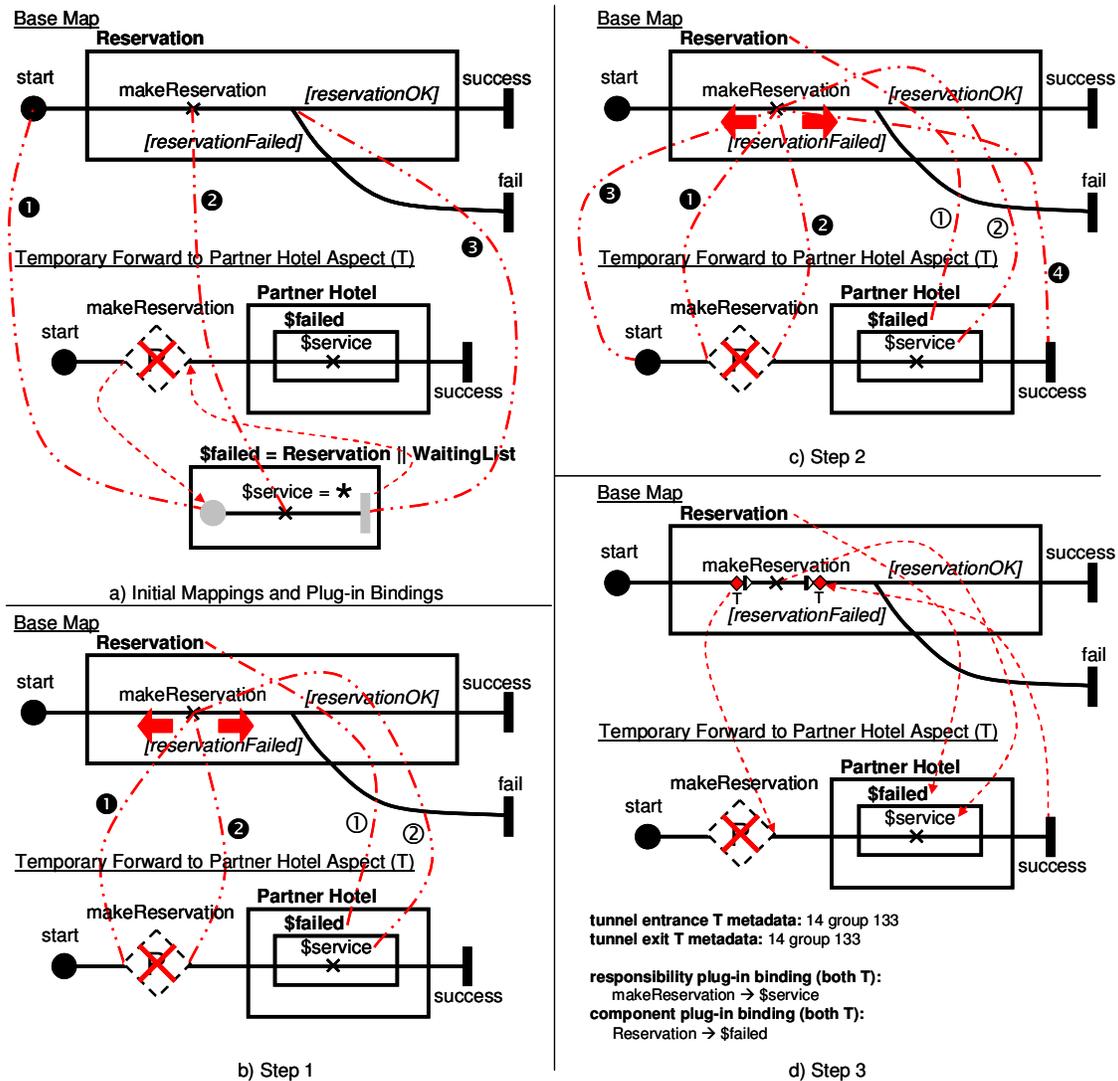


Figure 128 Composing AoUCM Concerns – Variables

In Step 3, the mapping is converted into a responsibility or component plug-in binding according to the type of the pointcut variable. Hence, mapping ① is converted into a component plug-in binding for all aspect markers, and mapping ② is converted into a responsibility plug-in binding. The AoView of the tunnel aspect marker is then able to substitute the variables in the aspect map based on the plug-in bindings.

Variables in the base model as opposed to variables in the aspect are another case that needs to be taken into account. If variables exist in the base model, they may not be matched unconditionally, but the match may depend on the actual values of the variables. In this case, a condition exists for the mapping of the base element that is a variable. The composition algorithm first assumes that the match is successful, adds the aspect markers therefore to the base model as usual, but then copies these conditions to the condition of the aspect markers added for the match. The conditions of mappings are collected in the preprocessing stage before Step 1 and are copied to the aspect markers in Step 3.

7.2.9 Insertion Points Before Start Points or After End Points

Aspectual properties can be inserted before a start point or after an end point, because named start or end points may be used on a pointcut map and are matched against start or end points in the base map, respectively. At Step 1 of the composition algorithm, named start points or named end points in pointcut maps receive special treatment when establishing the insertion point arrow and the mapping for the in-path or out-path of the pointcut stubs. First, the mapping of the in-path or out-path is the same as the mapping of the named start or end point, respectively. Second, the insertion point arrow of the mapped start point in the base map always points to the insertion point before the start point. Similarly, the insertion point arrow of a mapped end point always points to the insertion point after the end point as illustrated in Figure 129.

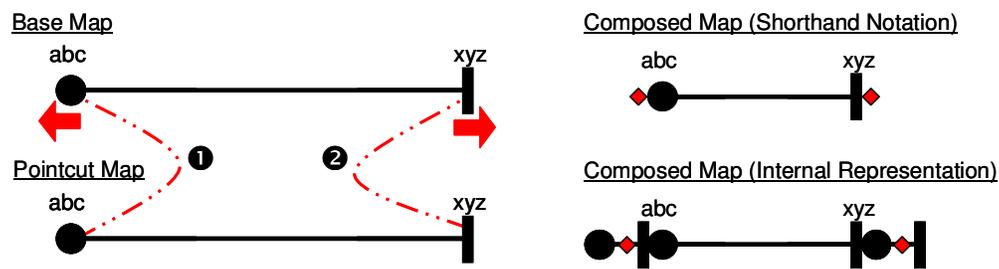


Figure 129 Composition with Named Start and End Points

Consequently, aspect markers have to be inserted in the base map either before the mapped start point or after the mapped end point (see shorthand notation in Figure 129). Aspect markers, however, cannot be simply inserted before a start point or after an end point as this would result in an invalid UCM model. Additional start and end points as shown in Figure 129 are required internally to ensure that the composed map is valid. While to the user only one start point or one end point appears, internally two start points or end points exist and any plug-in bindings of the mapped start or end point have to be transferred to the new start or end point, respectively (e.g., from abc to the first start point and from xyz to the last end point in Figure 129). This transfer ensures that the composed map can still be used properly as a plug-in map in the UCM model and that the traversal mechanism properly traverses the aspect and the start point or end point.

7.2.10 Conflicting Aspects

In general, the mappings of a match of one pointcut expression may overlap with mappings of a match established for another pointcut expression or even the same pointcut expression. This overlap is not a technical challenge for the matching algorithm as it deals with one potential match of one pointcut expression at a time. However, overlapping mappings may indicate conflicts between aspects.

A clear sign of possibly conflicting aspects is the transformation of the same join point in the base model. This possible conflict may be observed at Step 3 of the composition algorithm when an aspect marker is about to be added to the base model. If another aspect marker has already been added for the join point at the same insertion point during the same wave of compositions, a second aspect marker is not added (Figure 119). Instead, the aspect is added as a plug-in map to the existing aspect marker and plug-in bindings are specified for this plug-in map. This addition of a plug-in map may require changing the aspect marker from a static stub, i.e., the default setting, to a dynamic stub. The label of the dynamic aspect marker reflects each of the aspects represented by a plug-in map of the dynamic aspect marker.

The type of aspect marker may also have to be changed to indicate better the potential impact of all aspects combined. The tunnel entrance aspect marker is used only if all individual aspects require tunnel entrance aspect markers, indicating that the path

segment following the aspect marker is replaced in any case. The tunnel exit aspect marker is used only if all individual aspects require tunnel exit aspect markers, indicating that the aspect marker only defines the end of a path segment that is being replaced. A regular aspect marker is used if all individual aspects require regular aspect markers or the individual aspects require a mix of regular and tunnel exit aspect markers. A conditional aspect marker is used for all other cases, e.g., any mix of regular, tunnel entrance, and conditional aspect markers, to indicate that the impact of the individual aspects varies. A conditional aspect marker is not used for a mix of regular and tunnel exit aspect markers, because the path segment following the aspect marker is never replaced.

7.2.11 URN Model of Composed System

Given the results of the matching and composition algorithms, the AoURN model is easily transformed into a traditional URN model based on the AoViews. For AoGRL, an AoView is first created for each match of each pointcut expression. In a second step, all GRL model elements annotated with the pointcut deletion marker are removed from the model. Finally, all aspect-oriented annotations and all pointcut graphs are removed from the model, yielding a traditional GRL model. This approach allows for an easy visualization of the GRL model since the AoViews build on pointcut graphs with manually created layouts. A more advanced approach may want to perform the composition rules expressed by the pointcut graphs on the model directly without explicitly creating the AoViews. This approach, however, relies heavily for visualization of the composed result on the ability to automatically lay out GRL graphs, which is difficult to do well. The jUCMNav tool offers such a feature, which works in some cases but is not able to deal successfully with arbitrary GRL graphs.

For AoUCM, the AoViews are also first created for each match of the pointcut expression. Next, all pointcut stubs are removed, by replacing their in-paths with end points and their out-paths with start points while retaining the plug-in bindings of the pointcut stub for the new end and start points. Static aspect markers are then converted into regular static stubs. Dynamic aspect markers are also converted into a regular static stub that has a plug-in map expressing the flattened model of a dynamic aspect marker as described in Figure 119. Path segments that cannot be reached because, for example, a

[false] condition exists on a branch of an OR-fork could optionally be removed from the UCM model. Finally, all pointcut maps and annotations are removed from the UCM model with the exception of aspect marker groups.

Aspect marker groups need to be retained to indicate those situations where a scenario may enter a plug-in map through one stub and exit it through another stub, because this is not supported by the standard traversal mechanism of the UCM notation. To effectively support the transformation of AoURN models into standard URN models, the standard traversal mechanism must be extended to include Requirement 11f in Table 15 on Page 309.

Generally, the transformation of an AoURN model into a standard URN model may lead to a large number of additional URN diagrams with many duplicated model elements, if parameterized pointcut graphs and AoUCM variables are used. The alternative approach to apply the AoUCM composition rules directly on the model and then visualize maps on demand is not feasible because of complex layout issues. The auto-layout feature of jUCMNav is even less likely to produce a satisfactory result for UCM models than it is for GRL models.

7.3. Advanced Features of the Composition Algorithm

Special consideration has to be awarded to interleaved composition and the composition of semantics-based matching results. Interleaved composition affects all steps of the composition algorithm as described in Section 7.3.1, while semantics-based matching affects Step 1 and Step 3 as described in Section 7.3.2. These advanced features go well beyond typical before/after/around composition rules and allow the specification of potentially very complex composition rules.

The previous sections have shown that AoURN reliably composes already quite complex composition rules of aspects with one pointcut stub. The advanced features give even more power to the requirements engineer, which, on the other hand, needs to be used wisely. As discussed in Section 7.3.1, the AoURN language ensures that a large number of complex composition rules involving multiple pointcut stubs and various combinations of composition rules are addressed, but one can never be sure that all special cases have been resolved. Furthermore, the AoURN language specification cannot

anticipate the intentions of the requirements engineer. While Section 7.3.2 describes a composition environment that is consistent as much as possible, there may be cases where the intentions of a requirements engineer are not reflected by the composition. Consequently, the AoURN language specification is not a substitute for proper validation and verification procedures to guarantee that the desired end results are achieved.

7.3.1 Interleaved Composition

Interleaved composition has several unique features, setting it apart from other composition rules. The composition mechanism identifies interleaved composition by the existence of more than one pointcut stub on a single path of an aspect map. Pointcut expressions of interleaved compositions make use of pairs of connected, unnamed end and start points as shown in Figure 90 and Figure 130. The matching algorithm does not match these pairs against the base model as the pairs are not part of the pattern and are required only by the composition algorithm. However, the composition algorithm is much simplified if mappings for these pairs are established because the pointcut expression then appears as a series of regular pointcut expressions, i.e., a pointcut expression where each element is matched to a base element. The first regular pointcut expression exists from the first start point to the end point of the first pair, the second regular pointcut expression exists from the start point of the first pair to the end point of the second pair, and so on. The composition algorithm therefore adds a preprocessing stage before Step 1. For a start point in the pair, the preprocessing stage adds a mapping to the base element mapped to the path node preceding the pair (mapping ② in Figure 130.a). For an end point in the pair, the preprocessing stage adds a mapping to the base element mapped to the path node following the pair (mapping ③ in Figure 130.a). Given those modifications, Step 1 of the composition algorithm is then the same for interleaved composition as for any other composition rule.

The scanning in Step 2 of the composition algorithm also proceeds as before. The results of the scan, however, have to be interpreted in a slightly different way, because multiple pointcut stubs exist on an aspect map. On a valid aspect map, path segments may now exist between two pointcut stubs that do not contain a start point or an end point (e.g., the path segment between the Schedule Meeting and Meet Informant pointcut

stubs). Therefore, the composition algorithm does not issue an error and stop the composition, if no start or end point can be found for an in-path or out-path of a pointcut stub, respectively, as long as at least one out-path or in-path of a pointcut stub, respectively, is found instead. For example, scanning for start points of the in-path of Schedule Meeting in Figure 130 results in the pay start point. Scanning for end points of the out-path of Schedule Meeting, however, results only in the in-path of Meet Informant. Note that this situation also occurs for the special case in Figure 87.d on Page 133.

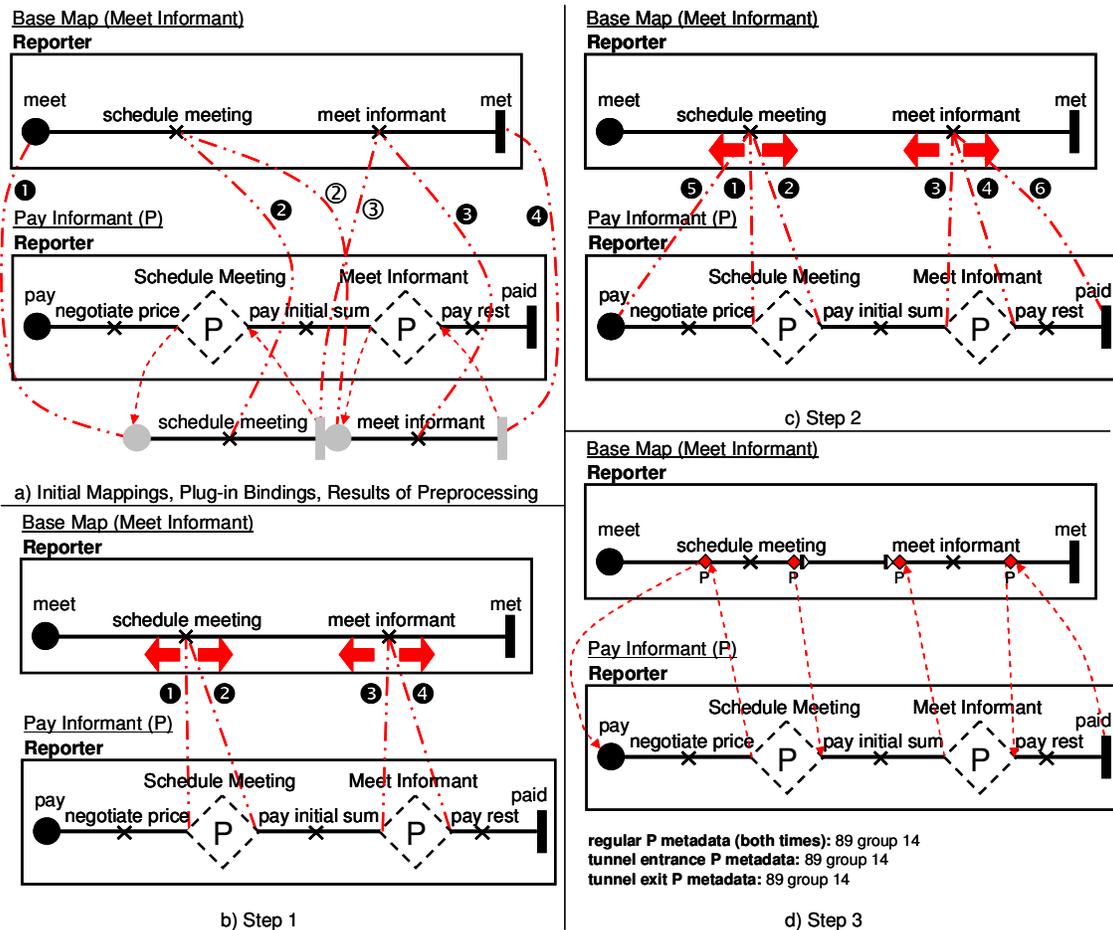


Figure 130 Simple Interleaved Composition

Furthermore, no mapping for a found out-path or in-path is established, because the mapping for it was already created in Step 1. For example, only the mapping for the start point found by the scan for the in-path of the Schedule Meeting pointcut stub in Figure 130.c needs to be created (mapping 5). No mapping, however, needs to be created for

the in-path of Meet Informant found by the scan for the out-path of the Schedule Meeting pointcut stub, because the mapping was already established in Step 1 (mapping ③).

Step 3 of the composition algorithm then proceeds as usual, except that for those in-paths and out-paths of a pointcut stub for which no start or end point, respectively, was found, only one plug-in binding with a tunnel aspect marker is established. For in-paths, a tunnel exit aspect marker is added, while a tunnel entrance aspect marker is added for out-paths (Figure 130.d). Consequently, interleaved composition always requires aspect marker groups because multiple aspect markers need to work together.

Finally, an optional optimization stage for the AoUCM model after Step 3 of the composition algorithm may merge a tunnel entrance aspect marker with a tunnel exit aspect marker including their plug-in bindings, if they follow each other immediately, as is the case in Figure 130.d. Also note that the interleaved composition respects the partial ordering of the two individual scenarios that are being interleaved. Changing the order of the scenarios is not allowed and may lead to undesired composition results.

Figure 131 and Figure 132 give examples that apply various composition rules at the same time. Figure 131 extends the previous example by allowing the price to be negotiated multiple times which also entails a rescheduling of the meeting. A loop is therefore added for these two activities, applying interleaving and loop composition rules at the same time. The difference to the previous example occurs in Step 2, where a) the scan of the in-path of the Schedule Meeting pointcut stub results in the start point and the out-path of the Schedule Meeting pointcut stub (hence the start point is still used for the second mapping) and b) the scan of the out-path of the Schedule Meeting pointcut stub results in the in-path of the Meet Informant pointcut stub and the in-path of the Schedule Meeting pointcut stub (hence still no new mapping needs to be established).

Figure 132 shows an example of the simultaneous application of the alternative and interleaved composition rules. The difference to the previous examples is that, in Step 2, the scans of the in-paths of both pointcut stubs establish a mapping with the start point s_2 , and the scans of the out-paths of both pointcut stubs establish a mapping with the end point e_2 (mappings ① and ③ for the start point and mappings ② and ④ for the end point in Figure 132.b). If these mappings are allowed, then the partial ordering constraint of interleaved composition is violated as parts of the scenarios are repeated out of

order. For instance, if the aspect marker after R1 is allowed to keep its out-binding, the scenario could exit this aspect marker. If the aspect marker before R2 is allowed to keep its in-binding to the start point, the scenario could enter this aspect marker after exiting the other aspect marker. Therefore, the responsibility R3 may be performed after already having finished the interleaved scenario, because the aspect marker after R1 has an out-binding with the end point, i.e., the scenario had already reached the end point and thus the end of the interleaved scenario.

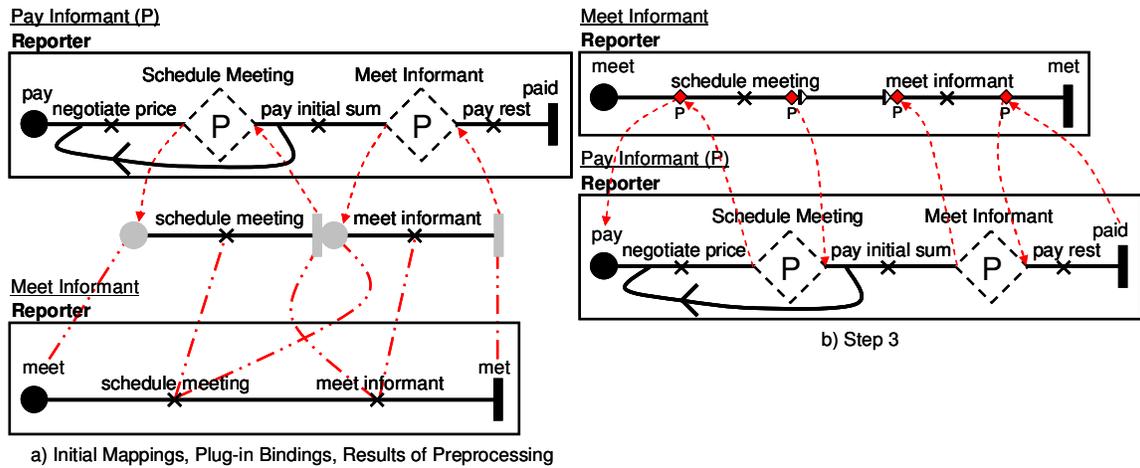


Figure 131 Interleaved and Loop Composition

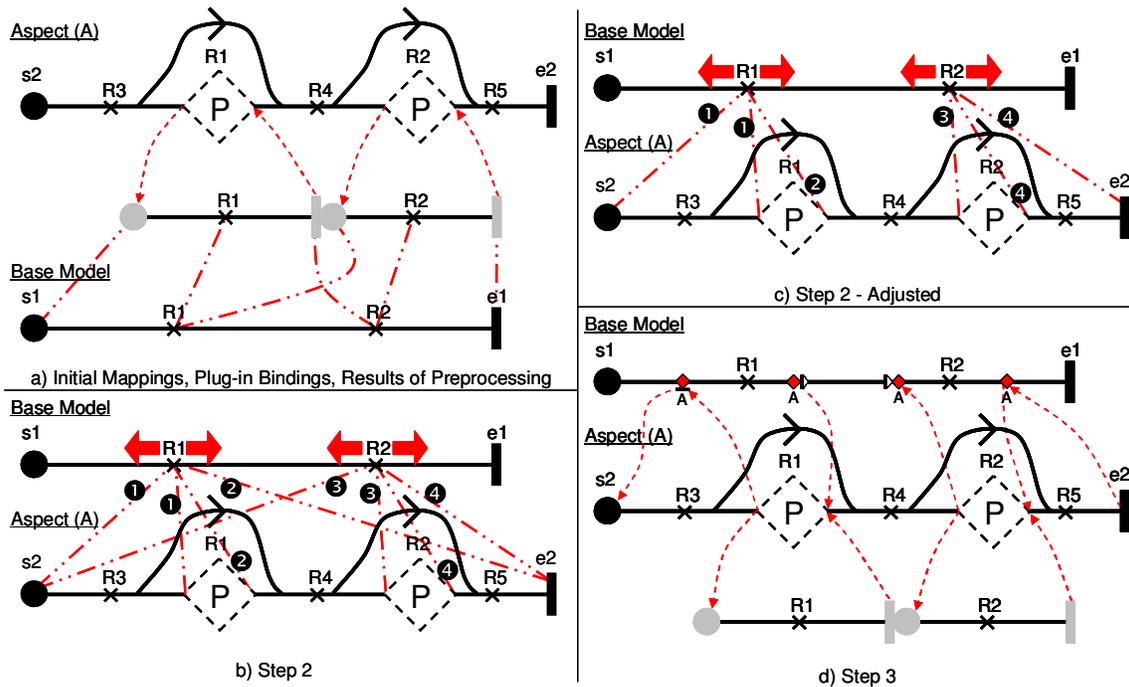


Figure 132 Interleaved and Alternative Composition

Hence, if two pointcut stubs claim the same start point, the one connected to the earlier path segment in the pointcut expression must be given the start point. Similarly, if two pointcut stubs claim the same end point, the one connected to the later path segment in the pointcut expression must be given the end point. Therefore, the mapping ❶ pair is kept because its pointcut stub links to the path segment with R1 which is before the path segment with R2 which belongs to the other pointcut stub with the mapping ❸ pair. Therefore, the second mapping ❷ with the start point is not kept (Figure 132.c). Similarly, the second mapping ❷ with the end point is also not kept because its pointcut belongs to the path segment before the path segment of the mapping pair that is kept, i.e., mapping pair ❹. The final result in Figure 132.d is then established as usual.

7.3.2 Composition of Semantics-Based Matching Results

The additional complexity introduced for the composition algorithm by results of semantics-based matching as compared to results of syntax-based matching lies in the fact that results of the former may span several levels of maps in a UCM model. While the beginning and the end of a matched pattern always reside on the same map for syntax-based matching results, they do not necessarily have to for semantics-based results but may instead cross map boundaries. This crossing of map boundaries further compounds the layout problem of the composed model. AoURN's approach to composition is advantageous in this case, because it does not require auto-layout, while other techniques for matching based on semantics do not consider layout issues at all (e.g., Klein *et al.* [72]).

For example, Figure 133 depicts two equivalent UCM models with five responsibilities each (R1 to R5). The first UCM model consists of only one map whereas the second is split up over three maps. Three of the five responsibilities are matched by the first pointcut expression in Figure 133.a. The aspect adds two responsibilities A1 and A2 before and after the matched elements, respectively. Hence, aspect markers are added before R1 and after R3 in both UCM models. The second pointcut expression in Figure 133.b contains two responsibilities and a stub. In this case, the aspect markers are added before R1 and after the stub on the second-level map for the second UCM model only. The UCM model without stubs is not matched even though it is semantically equivalent to the UCM model with stubs, because the modeler's decision to require a stub in the

matched pointcut expression takes precedence over matching based on semantics as explained earlier in the discussion of Figure 118 on Page 178. Given the mappings established by the matching algorithm, composition proceeds normally for the example in Figure 133.

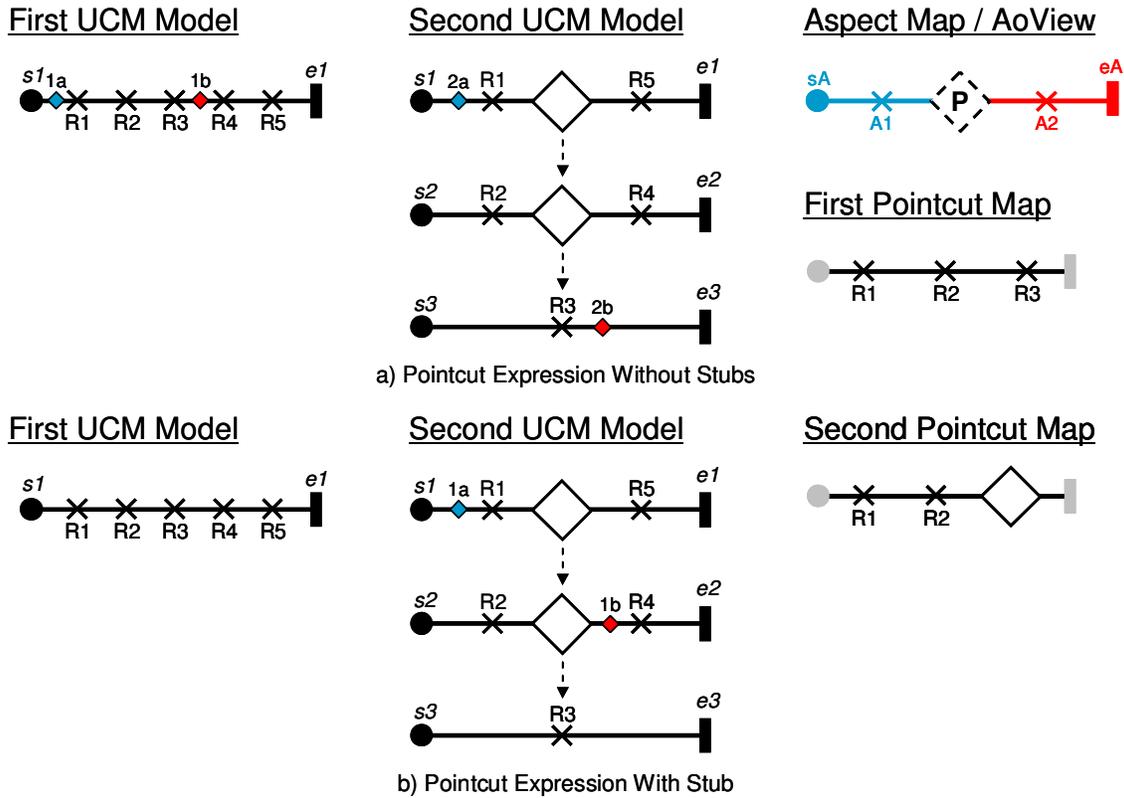


Figure 133 Composition for Semantics-Based Matching Results

Two problematic cases, however, exist for the composition of matches across map boundaries. For the first case, consider a UCM model such as the one in Figure 134 where two stubs share a common plug-in map. The pointcut expression, however, does not match the hierarchy of maps that includes the map with XYZ but only the hierarchy of maps that includes the map with R1. What happens when the traversal of the UCM model arrives at the bottom-level map from the map with XYZ? Should the scenario continue with the aspectual behavior or should the aspectual behavior be skipped? Furthermore, consider the situation where another aspect removes the blue aspect marker 1a altogether. Should the red aspect marker 1b still be available? Solutions could simply be to issue a warning to the requirements engineer or not match the pattern at all in this situation, argu-

ing that the UCM model does not actually reflect the pattern because implicit OR-joins and OR-forks exist for shared plug-in maps.

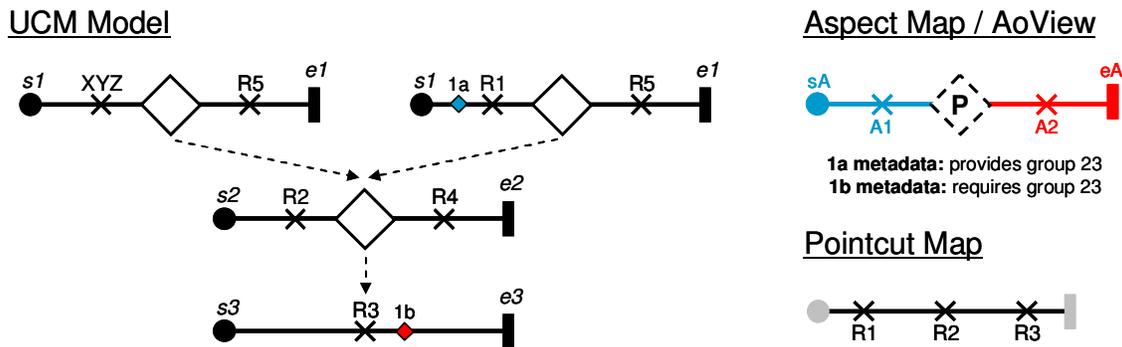


Figure 134 Composition with Shared Plug-in Maps

AoURN stipulates a more pragmatic solution that is based on the notion that an aspect map wants its aspectual properties to be applied as much as possible but only in the context of its matched pattern. This solution is consistent with regular matches that do not span multiple maps. Therefore, the red aspect marker 1b should be available only if the location of the blue aspect marker 1a has been visited. Then, however, another situation to consider is if the aspect does not add anything before the matched pattern, i.e., there is no blue aspect marker 1a. This situation is resolved by the AoURN composition mechanism, ensuring that the blue aspect marker 1a is always added, i.e., the blue aspect marker 1a links to an empty path segment before the pointcut stub on the aspect map.

A second situation to consider is if the blue aspect marker 1a is removed by another aspect. The red aspect marker 1b, however, should still be applied, unless of course explicitly removed. The context of the matched pattern is defined by the path defined by the three responsibilities R1, R2, and R3. AoURN ensures that this context is known when the red aspect marker 1b is reached.

This approach requires two changes to the composition mechanism. Similarly to replacements, empty segments are not ignored in Step 1 which ensures that all aspect markers are always added. Moreover, additional metadata needs to be added to the aspect markers in Step 3. All aspect markers connected to start points on the aspect map are tagged with the metadata name/value pair *aspect/provides group N*, with N being a unique number for this group of aspect markers. All remaining aspect markers on the aspect map (i.e., the ones not connected to start points) are tagged with the metadata

name/value pair *aspect/requires group N*. One can think of a scenario unfolding along the path with the scenario remembering that the first aspect marker has been visited. In addition, if an aspect marker with the metadata name/value pair *aspect/provides group N* is removed by another aspect, the metadata needs to be transferred to the other aspect in Step 3 to ensure that the scenario still detects that it unfolds along the path identified by the context of the matched pattern. This transfer requires a preprocessing stage before Step 1 that scans the match for aspect markers with such a tag.

The second problematic case occurs when a replacement pointcut stub is used instead of a regular pointcut stub. In this case, the location of the aspect markers are exactly the same but the UCM path traversal mechanism loses important contextual information during the traversal of the UCM model. Figure 135 illustrates what happens if a replacement pointcut stub is used. When the tunnel entrance aspect marker is reached in the UCM model, the traversal continues with the aspect. On the aspect map, A1 and A2 are traversed. Since the end point eA is connected to the tunnel exit aspect marker and the aspect markers belong to the same aspect marker group, the traversal continues with the bottom-level map. When its end point e3 is reached, the scenario should continue with the mid-level map because R4 used to be after R3 in the original model. Yet, the traversal mechanism is not aware of the mid-level map at this point, because it never reached the stub after R1.

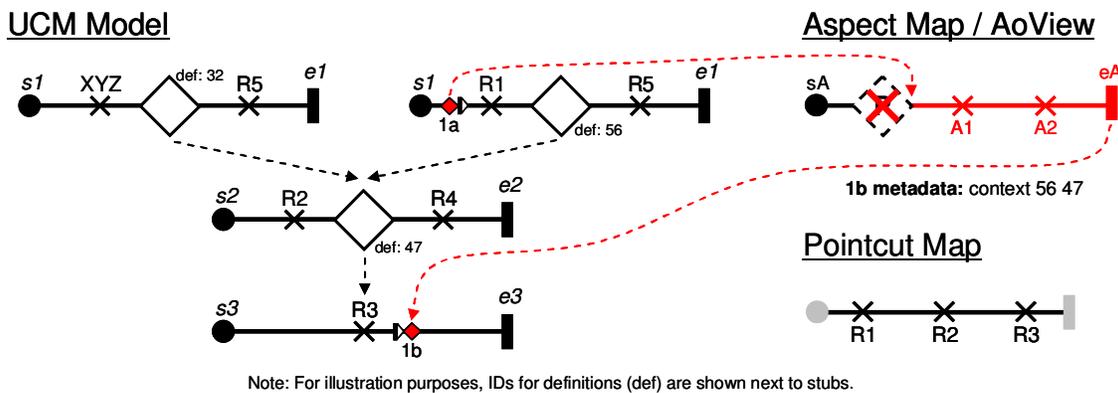


Figure 135 Composition with Lost Hierarchies

The mappings from the matching algorithm, however, indicate that a match spans all involved map levels. Therefore, the lost hierarchy information available in the matching result needs to be retained for the tunnel exit aspect marker. This is achieved by a pre-

processing stage before Step 1 and by tagging the tunnel exit aspect marker in Step 3 of the composition mechanism with the metadata name/value pair *aspect/context idList*. *idList* is a series of unique model identifiers of the stubs that defines the lost hierarchy. With this context information, the traversal mechanism adjusts the stack of visited maps – adding the second-level map to it – and is then able to continue traversing the UCM model as required.

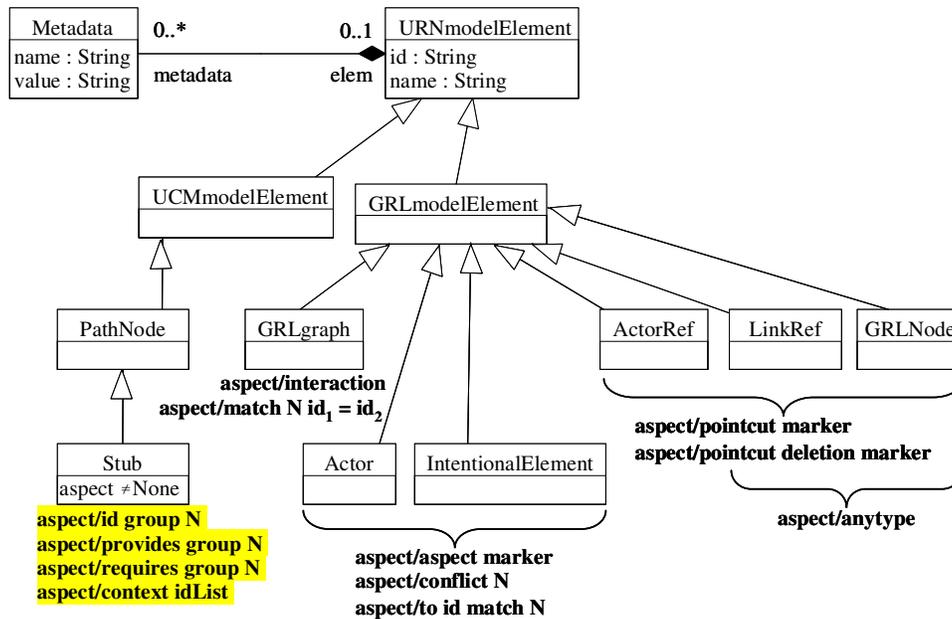


Figure 136 Updated Metadata for AoUCM Composition

Figure 136 shows the updated summary of all metadata required by AoURN including its AoUCM matching and composition mechanisms. The metadata name/value pairs *aspect/provides group N*, *aspect/requires group N*, and *aspect/context idList* may be assigned to a stub that is an aspect marker (i.e., the aspect attribute is not equal to None).

Extensions to the Traversal Mechanism – The traversal mechanism now needs to understand the new metadata information to ensure a) that an aspect marker is considered only if another aspect marker was visited before and b) that lost map hierarchies are recovered.

7.4. Beyond Matching and Composition

Given the results of the matching and composition mechanisms, several additional features can now be provided by an AoURN tool. For example, it is possible:

- to indicate for aspect A which other concerns (or diagrams or even model elements of these concerns) are transformed by A (this requires a simple examination of all aspect markers of A),
- to warn the requirements engineer if multiple aspects are transforming the same join point (this requires a simple examination of all aspect markers) and if one aspect is overriding another aspect,
- to warn the requirements engineer if mappings of different pointcut expressions overlap with each other (overlapping mappings from two pointcut expression may indicate a conflict or dependency relationship of the two concerns associated with the pointcut expressions),
- to warn the requirements engineer if a pointcut expression does not match anything in the base,
- to notify the requirements engineer of changes to the composition when the model is changed (this requires keeping track of the aspect markers at a certain point in time and then comparing them with the current aspect markers), and
- to warn the requirements engineer if the same pointcut is matched against a join point in multiple ways. For example, consider Figure 137 where a pointcut map contains an OR-fork with two branches. The OR-fork can be matched against the OR-fork in the UCM model in four ways as indicated. Therefore, the start point may be transformed four times, while each end point of the base model may be transformed twice.

A phenomenon that is called “emergent behavior” in the feature interaction community may also be observed when aspects and the base are composed together, possibly resulting in unexpected behavior. Undesired emergent behavior occurs when an aspect and the base work well individually but not when combined together. The feature interaction research carried out for URN does apply to AoURN and may be used to deal with conflicts caused by undesired emergent behavior. UCM scenario definitions may be used to describe preconditions and postconditions as well as start and expected end points of aspects and base behavior alike. After composing the system, scenario definitions may be interpreted to ensure that no preconditions and postconditions are violated and all expected end points are reached.

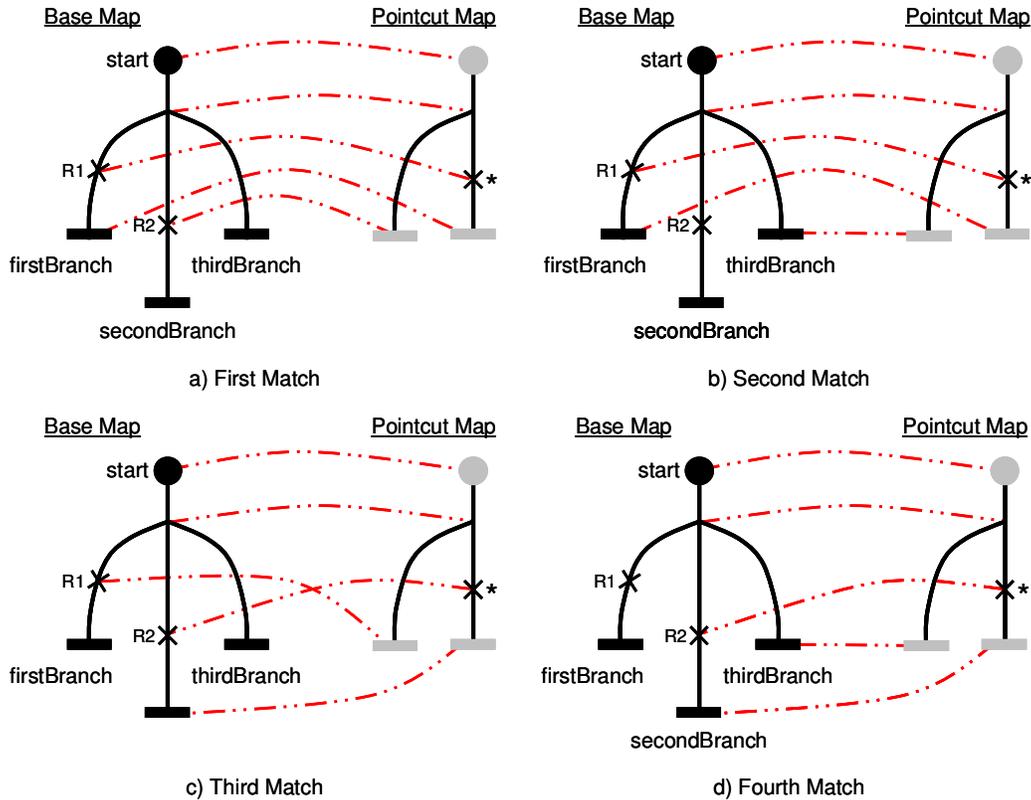


Figure 137 Multiple Matches of Pointcut Map

7.5. Summary

While Chapter 6 discusses the specification and visualization of AoURN models, this chapter presents the two-phase matching and composition mechanism of AoURN. The matching algorithm establishes mappings between elements in the pointcut expression and elements in the base model not only based on language syntax but also based on the semantics of the language. The composition algorithm deals with the insertion of aspect markers into the base model to facilitate navigation of the AoURN model in an aspect-oriented way with AoViews. Alternatively, the composition algorithm transforms the AoURN model into a standard URN model.

Composition occurs in waves based on the precedence rules defined by the concern interaction graph. In each wave, the concerns with the next highest priority are applied. Concerns with the same priority are first matched against the base model with the help of the matching algorithm (i.e., each concern is matched against a consistent base

model) before the composition algorithm changes the base model one concern at the time by inserting aspect markers based on the matching results.

Aspects that are applied at the same insertion point in the base model without precedence rules specified for them are addressed with dynamic aspect markers, a concept that is very similar to dynamic stubs but with a slightly different semantic interpretation. Instead of plug-in maps being executed in parallel as is the case with dynamic stubs, the plug-in maps of dynamic aspect markers are executed in random order. If precedence rules are specified in the concern interaction graph for such aspects, then a series of static aspect markers and dynamic aspect markers reflects the concern interaction graph.

AoURN's matching and composition mechanism supports interleaved composition rules that allow two scenarios to be combined without losing contextual information about either scenario. Furthermore, AoURN's matching algorithm supports pattern matches across UCM map boundaries and AoURN's composition algorithm can visualize both composed AoGRL and AoUCM models without having to resolve complex layout issues.

With the basic AoURN language framework now defined, the next chapters present the evaluation of this work which rests on three pillars. Chapter 8 investigates qualitative criteria that can be used to characterize aspect-oriented requirements engineering models and languages and assesses AoURN and other techniques against them. Chapter 9 then discusses quantitative measures and presents a case study involving two systems. Chapter 10 reports on the proof-of-concept implementation for AoURN in the jUCMNav tool.

Chapter 8. Qualitative Assessment of AoURN

Over the recent years, research emphasis has shifted more from Aspect-Oriented Programming (AOP) [68] to Aspect-Oriented Modeling (AOM) or Early Aspects (EA) [45] by investigating ways of addressing crosscutting concerns in requirements and design models. EA aims at managing concerns more effectively at these stages. Many approaches to Aspect-Oriented Requirements Engineering (AORE) are described in a recent survey [36], grouped into viewpoint, goal, scenario/use case, concern, and component-based approaches. Another recent survey targets AOM techniques for design activities [134]. As the Aspect-oriented User Requirements Notation (AoURN) is more closely related to the groups of goal-based, scenario-based, and use case-based AORE approaches, this chapter reviews these groups in more detail.

First, Section 8.1 presents a discussion of related work on aspect-oriented techniques for goal-oriented requirements models and contrasts them with AoURN. This discussion is followed by an assessment of aspect-oriented techniques for scenario-based requirements models in Section 8.3 which uses qualitative factors identified for scenario-based AORE approaches in Section 8.2 to compare AoURN with the other techniques.

8.1. Goal-Based Approaches to AORE

Gross and Yu [51] present an initial attempt at adding aspects to GRL. They introduce the concept of an intentional aspect as an abstraction for the design of aspects, the composition of actors and aspects, and linking aspects to implementation artefacts. Actors and intentional aspects encapsulate goals and alternative solutions of system modules and aspects, respectively. When actors and aspects are composed, their contributions to common non-functional requirements are merged. The main objective of the approach is to provide traceability between aspects and goals, and between aspects and implementation artefacts. However, composition of actors and aspects is not presently supported by tools.

Yu *et al.* [161] identify aspects in goal models based on relationships between functional and non-functional goals, propose goal aspects as a way to address scalability issues in goal models, and point out that the syntax of goal aspects requires further research. An example shows a textual definition of a goal aspect. AoGRL, on the other hand, uses a visual syntax for aspect-oriented goal models including pointcut expressions by employing a simple tagging approach. Niu *et al.* [107] build on the research by Yu *et al.* [161] for the purpose of tracing aspects from goal models to implementation and testing with the help of model transformations. Goal models are described with Q7, a textual language based on GRL. Goal aspects are expressed with extensions to Q7 that allow parameterized pointcuts to be specified. However, while the presented example encapsulates crosscutting relationships between tasks, Q7 still requires relationships between concerns at the goal level to be specified explicitly as contribution links. These links are then exploited by the pointcut expressions. AoURN, on the other hand, aims at fully encapsulating concerns in the goal model. Furthermore, the focus of Niu *et al.* [107] is on non-functional requirements only, while AoURN intends to also encapsulate scenario concerns in goal models.

Alencar *et al.* [2][3][4][5][6] identify aspects in i^* models and extend the notation to represent aspect-oriented concepts. Three rules are used to identify crosscutting concerns in i^* models. Based on the findings, the i^* model is restructured in an aspect-oriented way using a new notational element for aspects (a star). The extensions, however, do not allow crosscutting concerns to be fully separated from other concerns, and pointcut expressions are again defined in a textual way. AoGRL fully encapsulates crosscutting concerns in an aspect. Furthermore, Alencar *et al.* do not identify scenarios as aspects or concerns in i^* models whereas AoGRL does model scenarios as concerns because AoGRL is part of the larger AoURN context. Monteiro *et al.* [80] report on automated tool support for the identification of concerns as described in Alencar *et al.*'s work.

Gil [50] applies aspect-oriented techniques to goal models expressed with the KAOS [146] goal-oriented language by defining roles in aspect-oriented KAOS models that can be bound explicitly to elements in the KAOS model. Thus, aspectual properties are added to the model. The approach also identifies several relationships of aspects with elements to which aspects are applied (e.g., “contribute”, “guarantee”, “before”, “af-

ter”...). AoURN prefers parameterized pointcut expressions over explicit binding as the latter does not scale very well to large systems.

Niu and Easterbrook [105][106] focus mostly on discovering aspects in goal models with the Repertory Grid Technique and less on how to model discovered aspects. Consequently, their technique is orthogonal to AoURN and could be used with AoURN.

The techniques discussed so far in this section are limited to adding aspects to goal models, and do not focus on describing the operationalizations in goal models in more detail with other models such as scenario models. AoURN, on the other hand, considers goal models and scenario models at the same time. The remainder of this section discusses approaches that attempt to combine aspect-oriented modeling of goal and scenario models.

da Silva *et al.* [42][138] use aspect-oriented concepts to model goals based on the V-graph approach. V-graph also has capabilities for modeling scenarios, but these are limited and similar to ones offered by GRL by itself. The proposed composition rules are purely textual. AoURN, on the other hand, offers full support for scenario-based modeling with AoUCM and uses a fully visual approach for the modeling of aspects including their composition rules and pointcut expressions.

Kaiya and Saeki [63] propose a pattern-based technique to compose viewpoints. As in AoURN, they investigate goal and scenario models at the same time, but limit their composition technique for goal models to a simple combinatorial approach instead of more powerful pointcut expressions and composition rules as in AoURN. Their goal models are considered as AND/OR graphs and do not model dependencies and contributions like GRL. The proposed technique also augments use case diagrams with control and data dependencies, but nevertheless does not have the expressive power of AoURN’s scenario models. Use case models are transformed with the help of rules and patterns, which are defined rather informally. It is not clear whether the approach specifies all required information for the composition of goal graphs and use cases. Furthermore, every composition requires the specification of a crosscutting table which identifies explicitly all the elements that need to be composed together, leading to questions about the scalability of the approach.

Lee *et al.* [76] propose an aspect-enhanced, goal-driven approach that intends to model the interactions between aspects and the base with UML state machines and sequence diagrams. While AoURN allows the relationships of goals of various stakeholders to be modeled, Lee *et al.*'s approach is limited to associating goals with use cases. Use case diagrams are extended with annotations for goals as well as aspectual use cases and <<weave>> relationships that describe crosscutting in the use case diagram. Details of an aspectual use case are defined with the help of a textual specification template that explicitly identifies the base use cases and the specific join points in the base use cases to which the aspectual use case is applied as well as composition rules. Composition rules can be one of several operators covering insertion, concurrency, replacement, and the imposing of three types of constraints, i.e., duration, timing, and state invariant. The composition is further described with the help of an aspect-enhanced sequence diagram that specifies more precisely how the aspectual behavior is merged with the base use cases. The composition rules from the use case description are visualized as specialized combined fragments in the sequence diagram. A state machine gives another view of the behavior of the composition. In addition, aspect-enhanced sequence diagrams are also used to visualize the composed system. In contrast to Lee *et al.*'s approach, AoURN's composition rules are not limited to a specific set of supported compositions, but are exhaustive, allowing scenario models to be fully transformed. Furthermore, AoURN models systems in a more abstract way without reference to message and data details as well as states.

8.2. Qualitative Factors

The assessment of scenario-based modeling techniques for aspect-oriented requirements engineering goes even further than the review of goal-based AORE and is based on a set of qualitative factors for scenario-based AORE.

Exhaustive Composition: The ability of an aspect-oriented technique to express all required concern compositions, i.e., how limited is the usage of all modeling language constructs for composition purposes? Requirements models may be composed in many different, complex, and unexpected ways. A composition technique should be exhaustive in that it should provide the means to express all desired and frequently encountered composition rules. Considering frequently required composition rules for scenario-based

approaches, composition rules should therefore cover not just sequences, alternatives, and replacement (i.e., before/after/around) but make use of all constructs of the modeling language (e.g., concurrency, loops, and interleaving).

Scalability: An aspect-oriented technique should scale to large industrial models. In most cases, scalability means that it should be possible a) to modularize the requirements model into the right, manageable units and b) to parameterize composition rules in some way in order to cope with large numbers of similar compositions (quantification is one of the original tenets of aspect-orientation).

Reusability: An aspect-oriented technique should support the reuse of a concern's aspectual properties, composition rules, and pointcut expressions.

Familiarity: An aspect-oriented technique should employ a modeling notation that is already familiar to requirements engineers in order to ease adoption of the technique. If the chosen language is a visual language, then the aspect including composition rules and pointcut expressions should also be visual in order to avoid extensive switching (e.g., caused by using graphical and purely textual representations at the same time). It should also be easy to switch back and forth between traditional models and aspect-oriented models.

Formality: An aspect-oriented technique should be as formal as possible without becoming a barrier in practice.

At the right abstraction level: An aspect-oriented technique should be at the right abstraction level for requirements engineering. For an aspect-oriented technique to be effective, in particular for goal-based and scenario-based models in requirements engineering, the employed technique should be at an abstraction level where for example message or data details of interactions are not yet relevant.

Semantics vs. syntax: An aspect-oriented technique should take the semantics of its modeling notation into account when composing concern models to improve matching and composition results.

Handling of interactions: An aspect-oriented technique should be able to specify dependencies and conflicts between concerns and their resolutions. Furthermore, even though scenario models in requirements engineering are at a high abstraction level, an aspect-oriented technique should offer the ability to focus either on a single scenario view

but also on a view of the whole system that integrates many scenarios into a consistent overall picture to aid the detection of interactions.

Visualization of composition: An aspect-oriented technique should support requirements engineers in reasoning about the composed model by providing a way to visualize the effects of composition. Hence, the aspect-oriented technique should not require complex layout issues to be resolved to visualize the composed model.

Obliviousness: An aspect-oriented technique should allow concerns to be defined without changes to other concerns. Obliviousness is one of the original tenants of aspect-orientation but has since come under scrutiny as annotation-based or interface-based techniques have become popular. In general, however, a concern must not be heavily polluted by non-concern-specific information.

Tool support: Tool support should be available for an aspect-oriented technique, because it depends to a large extent on automation, e.g., when concerns are composed with the base model. Furthermore, tool support should allow aspect-oriented models to be specified by requirements engineers.

This list of qualitative factors is by no means an exhaustive list but it represents all factors that are considered for the comparison of scenario-based and use case-based approaches to AORE. Several other factors, mostly pertaining to usability and understandability issues, are not considered as a full usability and understandability study of the proposed notation is out of scope for this work. Furthermore, such studies are also not available for the techniques that are compared with AoURN. A thorough usability and understandability study for AoURN could allow an assessment of the ease of learning, the ease of adoption, and the intuitiveness of the notation, and would have to take the availability of tool support and training material into account.

8.3. Scenario/Use Case-Based Approaches to AORE

Several popular AORE approaches have been proposed for scenarios. In *Aspect-Oriented Software Development (AOSD) with Use Cases* [61], Jacobson and Ng view a well-written use case as a concern and add the notion of pointcuts to the traditional use case approach. A pointcut in one use case references extension points in other use cases in a textual way. An extension point identifies a step in the use case in which an extension

may occur. Furthermore, two new kinds of use cases are introduced – the “infrastructure” use case and the “perform transaction” use case – in addition to the traditional application use cases. An infrastructure use case describes scenarios required to address non-functional requirements. An infrastructure use case does not reference other use cases directly but it rather references extension points in generic perform transaction use cases. Such a use case models abstractly any type of interaction of an actor with the system, providing a generic description of the system. A perform transaction use case is eventually mapped to traditional use cases, effectively weaving aspect behavior described by infrastructure use cases into application use cases. This composition of concerns, however, is limited to extension use cases during use case modeling and to AspectJ-like constructs, i.e., before, after, or around, in later phases. Jacobson and Ng point out that aspects now allow use cases to be encapsulated throughout the entire software development lifecycle.

In *Scenario Modeling with Aspects* [151], Whittle and Araújo use UML sequence diagrams to describe non-aspectual scenarios and sequence-diagram-like interaction pattern specifications (IPSs) to describe aspectual scenarios. An IPS defines roles for classifiers, messages, and parameters. Binding the roles in an IPS to elements of sequence diagrams produces a composed system, which is then translated into state machines for validation. Alternatively, the sequence diagrams and IPS are first both translated into state machine representations (finite state machines and state machine pattern specifications (SMPSs), respectively) and then composed together at the state machine level [23] with the same binding technique. An SMPS is very similar to a state machine but also contains roles identified in an IPS. In both cases, the binding is specified textually and identifies explicitly elements to be bound. On the one hand, this binding allows for a very flexible composition that is as expressive as the modeling language itself. On the other hand, explicit bindings do not scale as well as parameterized bindings since each binding must be specified individually. Deubler *et al.* [44] propose another approach to aspect-oriented modeling of sequence diagrams that is however limited to simple before/after/around composition rules.

In the *Aspectual Use Case Driven Approach* [22][82], Moreira *et al.* propose to add extensions to UML use case and sequence diagrams in order to visualize how cross-

cutting non-functional requirements are linked to functional requirements expressed by use case diagrams or sequence diagrams. Non-functional requirements are captured with the help of templates. Moreira *et al.* [81] extend the set of use case relationships to include “constrain”, “collaborate”, and “damage” relationships while making use of activity pattern specifications (APSs). The new relationships describe how one use case impacts another, i.e., restricting it, contributing positively to it, or contributing negatively to it. An APS extends a UML activity diagram by allowing the specification of roles similar to IPSs [151] and SMPSs [23]. An APS is used to describe a use case in more detail. Various activity diagrams are composed by composition rules which are similar to the binding by Whittle and Araújo [151].

Sousa *et al.* [139] model crosscutting in use case diagrams caused by operationalizations required for non-functional requirements. The operationalizations are discovered with the help of the NFR Framework and the applied composition rules are limited to simple before/after/override/wrap rules. This work is still rather preliminary and is hence not taken into account for the comparison with AoURN.

Barros and Gomes [25] apply aspect-orientation to UML activity diagrams. The approach is based on an additional composition operation called activity addition, which allows the fusing of stereotyped nodes in one activity diagram with nodes in another. Stereotyping is effectively used as a pointcut expression, identifying explicitly nodes in another activity diagram for behavior merging. Zdun and Strembeck [140][162] extend UML activity diagrams with nodes for start and end points of concerns to model interdependent concern behaviors and to visualize concerns in a composed system with the help of concern-specific swimlanes. Another approach to aspect-oriented modeling of activity diagrams is described by Zhang *et al.* [164], who define aspectual properties and pointcut expressions in separate models. Cui *et al.* [41] also present an aspect-oriented approach for activity diagrams. This approach separates pointcut expressions and aspectual properties into separate activity diagrams identified by tags. A tag is also used to determine the join point in a pointcut expression to which the aspectual properties are applied through merging. Composition, however, is limited to sequential and concurrent composition rules and does not cover all constructs of activity diagrams. Furthermore, not all elements in activity diagrams are designated as join points, thus restricting the impact of aspects to

a select number of elements. On the other hand, variables allow matched base elements to be reused in the definition of aspectual properties. AoUCM is compared to the work of Cui *et al.*, because it is the most feature-rich approach out of the ones discussed in this paragraph.

Whittle *et al.* [152][153][154][155] propose a metamodel-based aspect composition technique (MATA) that uses graph transformation formalisms. This approach can be applied to any model for which a metamodel is defined, including UML state diagrams and sequence diagrams, for which tools and examples have been developed. MATA specifies aspectual properties, composition rules, and pointcut expressions in one diagram, allowing for exhaustive composition rules to be used but also making the aspectual diagram specific to the system to which it is applied, thus hampering reusability. If the aspectual properties of a MATA aspect are to be reused, they first have to be untangled from the composition rules and pointcut expressions. On the other hand, MATA employs critical pair analysis to reason about concern conflicts. Other generic weaving techniques that function on any metamodel are GeKo [83] and Kompose [47].

Rashid *et al.* [122] describe an approach for conflict identification and resolution for aspectual requirements. While the presented example uses a viewpoint-based approach to requirements engineering, it is argued that the technique can also be applied to other requirements engineering approaches including scenarios/use cases. Araújo and Coutinho [21] discuss aspects in a viewpoint-based requirements engineering approach that also includes use cases. Non-functional requirements, defined with templates, and use cases are linked to viewpoints. A use case that is included by another use case, that extends more than one use case, or that crosscuts several viewpoints is called an aspectual use case. This approach does not discuss composition of aspectual use cases with other use cases but focuses more on how to extend the work in Rashid *et al.* [122] for conflict resolution. Brito *et al.* [30] apply the Analytic Hierarchy Process (AHP) to reason about conflicting concerns that are again specified with the help of templates. The template specifies, among other things, a list of responsibilities the concern must provide and a list of contributions that indicate positive and negative influences of other concerns on the concern. Mehner *et al.* [78] enhance descriptions of use cases in activity diagrams with pre- and postconditions to aid in the detection of concern conflicts based on critical

pair analysis. While AoURN allows conflicts and dependencies to be captured in the concern interaction graph, it does not focus on conflict identification for aspects. Consequently, this is an orthogonal topic matter that could be combined with AoURN.

Another orthogonal topic matter is the identification of concerns. AoURN provides general heuristics to aid the requirements engineer but more sophisticated methods exist such as EA-Miner [130][131]. EA-Miner uses natural language processing techniques to detect crosscutting concerns in requirements documents.

In the UCM community, the applicability of UCM to model aspects was identified very early on by Buhr [32] but received little attention since then with the exception of work by de Bruin and van Vliet [43]. The approach by de Bruin and van Vliet is a top-down approach that adds a Pre stub and a Post stub for each location on a map that requires a change. The stubs allow behavior to be added before or after the location by plugging refinement maps into the stubs. Components on a map are identified by a Name:Type pair. A refinement map can be placed in a Pre or Post stub only if the component type on the refinement map matches the component type to which the Pre or Post stub is bound.

The techniques discussed in this section successfully address only some and not all of the qualitative factors identified in Section 8.2 and summarized in Table 6.

Only Whittle and Araújo, Whittle *et al.*, and AoUCM employ exhaustive composition mechanisms that can handle loops, concurrency, and interleaving in addition to more typical before/after/around composition rules. Jacobson and Ng as well as Moreira *et al.* do not address concurrency, loops, and interleaving, while Cui *et al.* support only sequential and concurrent composition and de Bruin and van Vliet support only before and after composition.

Most approaches are problematic in terms of scalability because composition rules identify targets of a composition only explicitly: Moreira *et al.* represent composition rules in an explicit and textual way without allowing parameterized expressions. Whittle and Araújo use textual and non-parameterized binding rules. de Bruin and van Vliet require the explicit addition of a Pre stub and a Post stub for each location on a map that requires a change. Jacobson and Ng, however, address scalability to a certain degree by modeling perform transaction use cases that capture generic interactions between an

actor and the system. Cui *et al.*, Whittle *et al.*, and AoUCM each supports parameterized pointcut expressions. Note that scalability of a technique is a very complex qualitative factor to be assessed with a simple binary yes/no scale.

Table 6 Comparing Scenario-Based Approaches to AORE

	Exhaustive	Scalable ¹⁾	Reusable	Familiar ¹⁾	Formal	Abstraction Level	Semantics vs. Syntax	Handling Interactions	Composition Visualization	Oblivious	Tool Support
Jacobson and Ng [61] (AOSD with Use Cases)		✓	✓	✓	2)	✓		3)	✓		✓
Whittle and Araújo [23][151] (Scenario Modeling with Aspects)	✓			✓	2)					✓	
Moreira <i>et al.</i> [22][81][82] (Aspectual Use Case Driv. Appr.)					2)				✓	✓	
Cui <i>et al.</i> [41] (UML Activity Diagram)		✓	✓	✓	2)	✓				✓	
Whittle <i>et al.</i> [153][154][155] (Graph Transformation – MATA)	✓	✓		✓	✓	✓		✓		✓	✓
de Bruin and van Vliet [43] (Quality-Driven Sw. Arch. Comp.)			✓	✓	2)	✓			✓		✓
Aspect-oriented Use Case Maps (AoUCM)	✓	✓	✓ ⁴⁾	✓	2)	✓	✓	✓	✓	✓	✓

1) Hard to assess on a binary yes/no scale.

2) Qualifies only as a semi-formal composition technique compared to graph transformations.

3) Limited support is available.

4) Applies only to AoUCM; AoGRL has similar issues as MATA.

Many techniques do not clearly separate pointcut expressions and aspectual properties, making it thus more difficult to reuse aspectual properties. This applies to Moreira *et al.*, Whittle *et al.*, and Whittle and Araújo. Note that AoGRL also falls into this group. Jacobson and Ng, however, tackle reusability through the definition of generic infrastructure use cases and perform transaction use cases. Each of Cui *et al.* and AoUCM clearly separates pointcut expressions from aspectual properties, ensuring that an aspect can be applied to a new model by simply defining or plugging in, respectively, a new pointcut expression. AoUCM also allows pointcut expressions to be reused for different aspects. de Bruin and van Vliet support the reuse of behaviour by plugging generic maps into Pre stubs and Post stubs. Note that the concept of dynamic stubs could be applied to the graph transformation approach of Whittle *et al.* to eliminate the intertwined specification of aspectual properties and pointcut expressions.

Moreira *et al.* require several extensions to UML diagrams to visualize aspects. All other techniques, however, use familiar modeling languages. In addition to scalability, familiarity is another difficult qualitative factor to judge for a technique.

Compared to the rigor and formal foundation of a graph transformation, every other technique qualifies at the most as a semi-formal technique.

AoUCM is at a higher level of abstraction than the work by Whittle and Araújo, which is at the message or state machine level. Moreira *et al.* make use of some models that are at the same level of abstraction as AoUCM and some models that are at a lower level of abstraction than AoUCM. Every other approach is at the same abstraction level as AoUCM.

None of the approaches except for AoUCM takes the semantics of the modeling notation into account when composing concerns. AoUCM defines semantic equivalence classes that contain syntactic constructs that have the same meaning, focusing especially on hierarchical structuring. AoUCM's matching and composition mechanism then takes these equivalence classes into account. Therefore, refactoring operations can be safely performed on the AoUCM model without the risk of breaking some of the pointcut expressions.

None of the techniques handle interactions of concerns well except for AoUCM and Whittle *et al.*'s MATA. AoUCM specifies concern interaction graphs that capture dependencies, conflicts, and resolutions based on precedence rules. MATA also allows for the definition of precedence rules and supports critical pair analysis for the detection of undesired interactions. Jacobson and Ng have some limited support for handling interactions between extensions use cases applied at the same extension point.

Approaches based on use cases (Jacobson and Ng, Moreira *et al.*) do not face complex layout issues when visualizing the composed model, since use case diagrams do not tend to be overly complex. However, techniques based on sequence diagrams (Whittle and Araújo), activity diagrams (Cui *et al.*), or graph-based transformations (Whittle *et al.*) do have to deal with such issues. The addition of stubs, however, is manageable in de Bruin and van Vliet's approach. AoUCM's usage of AoViews circumvents the whole problem altogether by using views for the visualization that have already been provided by the modeler. Note again that the concept of dynamic stubs could be applied to tech-

niques based on sequence diagrams, activity diagrams, or graph-based transformations to ease the visualization of the composed system.

While most techniques allow an aspect to be defined in a way that ensures that other concerns are oblivious to the existence of the aspect, Jacobson and Ng add extension points and de Bruin and van Vliet add stubs directly to the base model, causing it to be relatively polluted with concern-specific information.

Jacobson and Ng make use of UML tools for their approach for the specification of aspect-oriented models. Specific tool support for automated composition is not required because aspect-oriented models in Jacobson and Ng's approach are composed explicitly by requirements engineers. Tool support is available for the MATA technique for sequence diagrams and state machines, both in terms of model specification and model composition. The jUCMNav tool offers an AoURN modeling environment that includes support for the specification of aspect-oriented models and for their composition. The approach by de Bruin and van Vliet is also supported by the jUCMNav tool, because no automated composition is required as models are composed explicitly by the requirements engineer, which is supported by jUCMNav. For all other techniques, tool support for composition is required but not available.

8.4. Summary

Generally, AoURN has been influenced by the concepts and ideas of a) symmetric approaches to aspect-oriented modeling that advocate a separation of aspectual properties, pointcut expressions, and composition rules, b) MATA, which uses a tagging approach that is also used for GRL as it is appropriate for highly interconnected goal models, and c) AspectJ in terms of how to indicate composition to the requirements engineer. AoURN's aspect markers are reminiscent of AspectJ's annotations of source code in the vertical bar to the left of the code editor.

Many techniques for aspect-oriented goal and scenario modeling are summarized in this chapter, but none except for AoURN combines goals and scenarios in one framework and is supported at the same time by an exhaustive aspect composition technique appropriate for the model type to which it is applied. Furthermore, AoURN is the only technique that employs a matching algorithm that is enhanced based on semantics. While

AoURN is built on an international requirements engineering standard, all other discussed techniques based on UML are also built on an international standard. Similarly, AoURN belongs to a select group of techniques for which tool support is provided, aspectual properties and patterns may be reused without adaptation, and interactions are handled. In terms of the formality of the approach, AoURN rivals most other techniques except for the graph transformation-based approach by Whittle *et al.*, which is superior in this regard.

Chapter 9. Quantitative Assessment of AoURN

This chapter describes the analysis of AoURN's capabilities compared to those of traditional URN models in terms of scalability, modularity, reusability, and maintainability. The evaluation of AoURN vs. traditional URN is based on a quantitative assessment of the structure of the models built with AoURN and URN. This is therefore an experiment that must compare equivalent models, i.e., the content of the models must be the same. The evaluation looks at the structure of two equivalent AoURN and URN models and determines characteristics of these models with the help of metrics adapted from literature to assess modularity, reusability, and maintainability [133]. In addition, a task-based evaluation assesses the impact of common update tasks on the AoURN and URN models.

The metrics are applied in a case study involving two systems. The Car Crash Crisis Management System (CCCMS) is chosen because it has been posed as a large challenge problem by the aspect-oriented modeling research community [69]. The second is the Your Key Knows (YKeyK) system already presented in Section 6.8. This system has been used to describe aspect-oriented models before [2]. The findings of this case study have also been reported previously for CCCMS [95] and for YKeyK [96].

The experiment does not attempt to determine which technique may define a more complete model. Nevertheless, there is anecdotal evidence that focusing on concerns helps make a model more complete [96]). The setup of the CCCMS challenge problem does not easily allow such an experiment since the content of the CCCMS document is deemed to be complete and must not be substantially altered. YKeyK, on the other hand, is a smaller example in which completeness is not as significant an issue. Furthermore, this assessment is also not a usability study, which would require groups of users trying to independently model the CCCMS or YKeyK with AoURN and URN. Undeniably, such an experiment will have to contend with several issues in addition to the core URN and AoURN language features. Issues such as available tool support and sufficient user training have to be considered and influence the experiment's result to a certain degree.

While these experiments would also answer important questions about URN and AoURN and hence should be undertaken in the future, they are out of scope, as this work is, for now, more focused on assessing the core language features of URN and AoURN with the help of metrics.

First, the general AoURN modeling process is presented in Section 9.1, discussing also how the process had to be adapted for this case study. Section 9.2 presents a selection of the models for the CCCMS, while Section 9.3 briefly reiterates the YKeyK system. Section 9.4 explains the metrics used for the assessment and Section 9.5 reports on the comparison of URN and AoURN.

9.1. Overview of the AoURN Modeling Process

This section presents a general modeling process for AoURN models. The highly iterative process consists of five process modules.

P1) Identify concerns – Concerns are identified using the guidelines AoURN provides for concern identification, i.e., it treats all use cases, NFRs, and stakeholders as concerns.

P2) Build scenario model with AoUCM – Behavioral aspects of the available system documentation are assessed against the identified concerns and structured into an AoUCM model.

P3) Describe NFRs with AoUCM – Non-functional concerns are layered over the functional concerns, by modeling NFRs described in the available system documentation with AoUCM.

P4) Model stakeholder dependencies and impact with AoGRL – The dependencies between stakeholders are modeled in a goal model with AoGRL. Furthermore, the impact of the functional and non-functional concerns on the stakeholder goals is modeled with AoGRL.

P5) Capture concern dependencies and conflicts – Whenever a new concern is being modeled, the dependencies and conflicts with other concerns are captured in the concern interaction graph.

Adaptations to the Process for the Case Study

Given that the evaluation of AoURN also requires a non-aspect-oriented URN model to be sufficiently defined for comparison purposes, the AoURN modeling process is slightly adapted. In P2, the use cases described in the documentation are modeled in a non-aspect-oriented way with UCM. The UCM model is then assessed against the identified concerns and restructured into an AoUCM model. This assessment is equivalent to assessing the use cases against the identified concerns, as there is a straightforward, almost one-to-one mapping between the use cases and the UCM model. However, assessing the UCM model directly ensures that the resulting UCM and AoUCM models are consistent.

In P3, the UCM model for the NFR concerns is derived from the AoUCM model to ensure consistency between the two models. This decision is motivated in more detail at the beginning of Section 9.2.5. In P4, the stakeholders are also modeled with GRL. This is necessary for a comparison of standard URN and AoURN models after the completion of all process modules. As the comparison is based on metrics adapted from the literature [133] and a task-based evaluation, the URN and AoURN models have to be consistent and cover the exact same content to avoid skewed results of the applied metrics. Any chosen modeling process for this comparison has to ensure that the two models are equivalent, either through a separate process step that balances out the two models after both have been built separately by the same team or different teams, or by taking both models into account during their construction. The assessment presented here does the latter.

9.2. Car Crash Crisis Management System

The Car Crash Crisis Management System (CCCMS) is responsible for a coordinated response to crisis situations following a car crash. It helps identify, assess, and handle a crisis situation by facilitating the communication between all involved parties, ranging from ambulances to tow truck drivers. Furthermore, the CCCMS manages all required resources and provides timely access to relevant crisis-related information. The modeling process described in Section 9.1 resulted in 25 non-orthogonal concerns of the CCCMS being modeled with AoURN.

9.2.1 Identification of Concerns (P1)

AoURN provides only rough guidelines for identifying concerns as its strengths lie in the specification of concerns. AoURN is, therefore, ideally to be combined with other concern identification methods such as EAMiner [131], which automatically identifies concerns. As the purpose of this case study is to focus on AoURN alone, AoURN's guidelines are followed. Therefore, use cases, NFRs, as well as stakeholders are considered as concerns and iteratively refined. The following 14 non-orthogonal concerns are modeled with AoURN. These concerns are further broken down into sub-concerns, resulting in a total of 25 individual sub-concerns [35].

The four major functional concerns that are identified are Recommend Strategies (recommend strategies based on the crisis and resource availabilities), Resource Management (assignment and release of resources such as first aid workers, ambulances, and fire trucks), Coordination (execution, adaptation, and monitoring of missions), and Communication (flow of crisis/mission information between actors and the system as well as communication infrastructure). Nine of the ten use cases in the CCCMS document are sub-concerns of these four major functional concerns:

- Coordination: Resolve Crisis, Execute Mission, Execute Super Observer Mission, Execute Rescue Mission, Execute Helicopter Transport Mission, Execute Remove Obstacle Mission
- Communication: Capture Witness Report
- Resource Management: Assign Internal Resource, Request External Resource

Note that each use case is assigned only to one concern in the above list, i.e., its main concern. Other concerns may still crosscut a use case (e.g., the Communication concern appears frequently in use cases assigned to the Coordination concern).

Of the eleven NFRs listed in the CCCMS document, six are modeled as concerns. These are: Persistence, Statistical Logging, Availability, Security (with its three sub-concerns Authentication, Access Control, and Encryption), Mobility (with its three sub-concerns Mobile Infrastructure, Location Status, and Map Information System), and Safety (with its three sub-concerns Weather Information System, Emissions Monitoring System, and Criminal Activity Monitoring System). Note that the tenth use case in the CCCMS document belongs to the Security NFR concern.

Finally, a large number of stakeholders are mentioned in the CCCMS document (Coordinator, Super Observer, CCCMS Employee, External Worker, Phone Company, Surveillance System, System Admin, Witness, Police, Fire Department, Hospital, Doctor, Ambulance, Victim, First Aid Worker, Helicopter Pilot, and Tow Truck Driver) and many more are not mentioned (e.g., Funding Agency, Government, as well as the General Public). Out of these, the following four stakeholder concerns are modeled: the Coordinator, the General Public, the Resource (representing all resources such as Super Observer, Police, Fire Department, etc.), and the Government (representing the customer of the CCCMS).

Due to the lack of details relevant to URN/AoURN goal modeling in the available documentation about the proposed CCCMS, the CCCMS models are limited to four stakeholders. GRL is a technique to capture and evaluate alternatives for reaching stakeholder goals. The CCCMS document does not describe well such alternatives and their rationales nor does the document describe well stakeholders and their goals. Four stakeholders are a) a large enough number to demonstrate the capabilities of GRL and AoGRL and b) small enough to minimize the amount of information required for goal modeling that is not explicitly stated in the CCCMS document but must be derived implicitly from the CCCMS document with the help of common knowledge.

For example, the CCCMS document does not specifically state which NFRs are of interest to which stakeholders, what the dependencies are between stakeholders, and what the priorities of a stakeholder's goals are. Instead, the general objectives listed at the beginning of Section 2 of the CCCMS document, the goals mentioned in the description of the non-functional requirements, as well as the goals of the use cases must be correlated with the stakeholders and further refined based on common knowledge.

Similarly, since the CCCMS document does not provide information about the advantages and disadvantages of possible solutions, the existing crisis management system is considered as sole alternative to the CCCMS. According to the CCCMS document, the existing, non-computerized system manually keeps track of crisis information and does not recommend strategies for dealing with the crisis.

Given the concerns identified in this section, crosscutting is expected to manifest itself in general between the following concerns. The Coordination concern is crosscut by

the three other main functional concerns and their use case concerns. The Coordination concern, on the other hand, slightly crosscuts the Communication concern. Furthermore, the Coordination concern represents the core functionality of the CCCMS. It is an essential requirement while the other functional concerns could be reduced or removed from the CCCMS. The NFR concerns crosscut several use case concerns and thus the four major functional concerns. In addition, the NFR concerns may crosscut each other. The use cases and thus the four major functional concerns crosscut the stakeholder models as a single use case may impact the goals of several stakeholders. Finally, the NFR concerns also crosscut stakeholders and use cases as an NFR may be of interest to many stakeholders in the context of many use cases.

Since the content of the CCCMS models that are to be built is given by the CCCMS document, the most influential decision made by the modeler is the choice of concerns. A different choice of concern could lead to vastly different and diverse models. However, once the concerns are defined, the creation of the AoURN and URN models given a static CCCMS document becomes rather mechanical. The biggest challenge is to comply with the description of the problem in the CCCMS document and to consistently assign modeling elements to the correct concerns. Therefore, major design decisions are not expected for the remaining process steps.

9.2.2 Model Functional Concerns with UCM (P2 – modified)

The textual use cases in the CCCMS document are first translated into a UCM model. Figure 138 and Figure 139 show the Resolve Crisis use case as an example of the UCM model derived from the use cases in the CCCMS document. The UCM model of the Resolve Crisis use case shows that after capturing a witness report, the system recommends missions to be selected by the coordinator. The selected missions are executed, which is shown in more detail on the ExecuteMission plug-in map. Once a mission has finished, the mission file is updated and if all missions have been executed, the crisis is resolved. The second out-path of the ExecuteMission stub leading back to recommendMission(s) indicates that if there are problems with the current mission, the use case continues with the system recommending missions based on current crisis information. In addition, the AND-fork after selectMission(s) shows parallel activity. If new information is available

at the newCrisisAndMissionInfoAvailable start point, the waiting place is triggered and the coordinator assesses the new information. If it is deemed that changes are required to the current missions, then the use case continues with the system recommending missions. If not, the system continues to wait for new information.

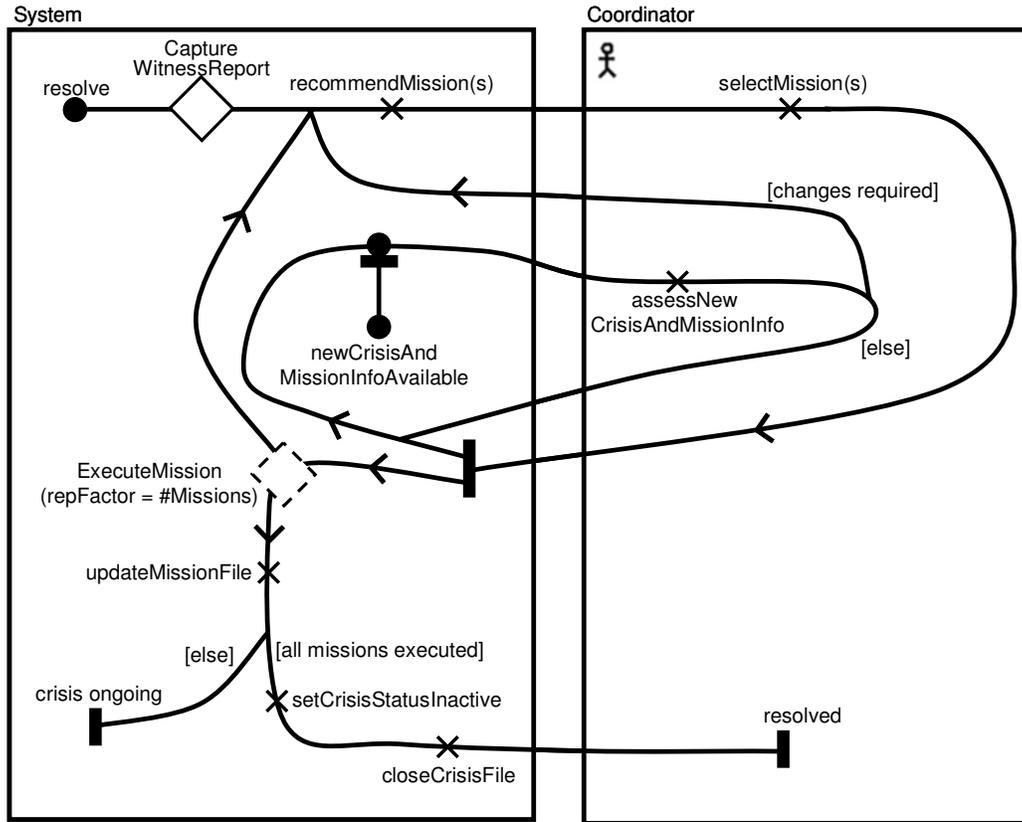


Figure 138 UCM: Resolve Crisis Use Case

A replication factor is defined for the plug-in map (Figure 139) of the ExecuteMission stub in Figure 138, indicating that as many missions (i.e., instances of the plug-in map) are started in parallel as selected by the coordinator.

The plug-in bindings of the ExecuteMission stub specify that when the path enters the stub in Figure 138, the use case continues at the execute start point of Figure 139. It ends either at the executed end point of Figure 139, exits the stub at its bottom out-path, and hence continues on to updateMissionFile in Figure 138, or it ends at the resource not available/more missions required/mission failed end point of Figure 139, exits the stub at its top out-path, and hence loops back to recommendMission(s) in Figure 138. In Figure 139, internal and external resources are first requested which may fail (RequestRe-

source(s) stub; again, replication factors define the number of requested internal and external resources). If an internal resource request fails, it may be substituted by an external resource request (substituteInternalWithExternalResource responsibility). Upon successful resource allocation, the system assesses whether the resource needs to be transported by helicopter to the crisis location (TransportWithHelicopter stub). If yes, a new mission for the helicopter pilot is initiated. In any case, the resource confirms arrival at the crisis location (ConfirmArrival stub), then performs its mission (ExecuteResourceMission stub), before confirming departure (ConfirmDeparture stub). In parallel to this, the resource may receive updated mission information (UpdateMissionInfo stub). The ExecuteResourceMission stub is a dynamic stub because each resource's mission may be very different from other missions and therefore requires its own plug-in map (e.g., the super observer, rescue, helicopter transport, and the remove obstacle missions). Finally, a mission report may or may not be submitted by the resource before the resource is released.

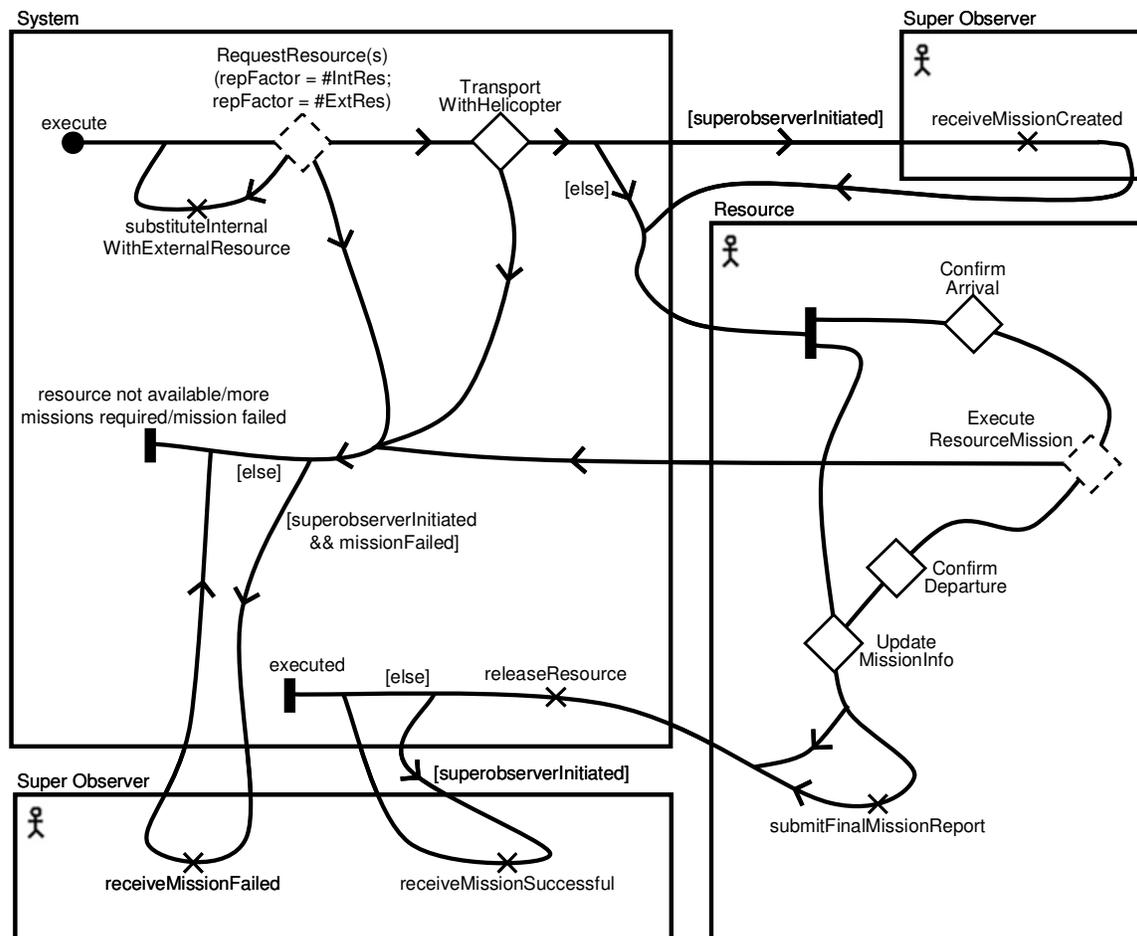


Figure 139 UCM: Resolve Crisis Use Case – ExecuteMission Plug-in Map

In addition, if the mission was initiated by the super observer, then the super observer is notified about the creation, success, and failure of the mission.

9.2.3 Model Functional Concerns with AoUCM (P2)

For consistency reasons, the UCM model is restructured into an AoUCM model based on an assessment of the whole UCM model against the functional and non-functional concerns identified in Section 9.2.1. Table 7 lists which map elements of the use case in Figure 138 and Figure 139 belong to which concerns. While the Coordination concern is identified as the main concern for the use case, large portions of it can be attributed to five other concerns.

Table 7 Assessment of Resolve Crisis UCM Model Against Identified Concerns

Concern	<i>Use Case (UCM):</i>	
	Resolve Crisis (ResolveCrisis map)	Resolve Crisis (ExecuteMission map)
Recommend Strategies	recommendMission(s)	
Resource Management		RequestResource(s), substituteInternalWithExternal-Resource, releaseResource
Coordination	✓	✓
Communication	CaptureWitnessReport, assessNewCrisisAndMissionInfo (including the AND-fork, waiting place, and starting point)	ConfirmArrival, ConfirmDeparture, Update Mission Info, submitFinalMissionReport
Super Observer Mission		receiveMissionCreated, receiveMissionSuccessful, receiveMissionFailed
Helicopter Transport Mission		TransportWithHelicopter

✓ A checkmark indicates that the map belongs to the concern, including all map elements except for those listed in the same column under a different concern.

The Communication concern is deemed to deal with the flow of crisis/mission information among the actors and, therefore, includes ConfirmArrival and ConfirmDeparture, which indicate that an assigned resource has arrived or departed, respectively. The Resource Management concern, on the other hand, just deals with the assignment and release of resources. For similar reasons, assessNewCrisisAndMissionInfo belongs to the Communication concern instead of the Coordination concern. The Coordination concern deals only with execution, adaptation, and monitoring of missions, but not with the flow of information.

A restructuring of the UCM model for the Resolve Crisis use case according to aspect-oriented principles is expected to better encapsulate the four major functional concerns as well as the Execute Super Observer Mission and Helicopter Transport Mission sub-concerns. Figure 140 and Figure 141 show the AoUCM model of the Resolve Crisis use case with all tangled concerns removed, leaving only the Coordination concern.

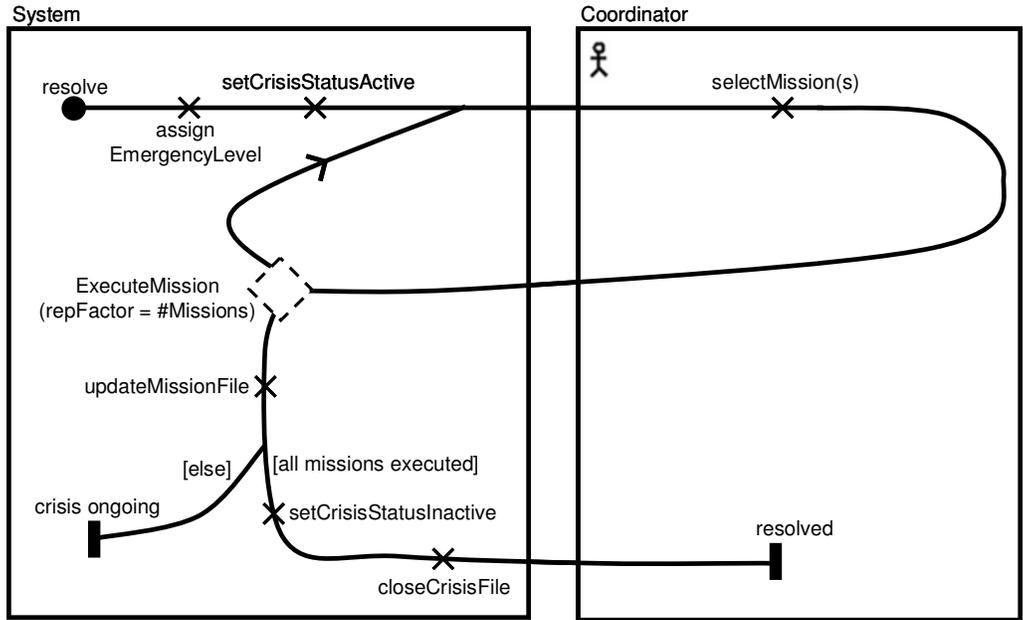


Figure 140 AoUCM: Coordination Concern – Resolve Crisis Use Case

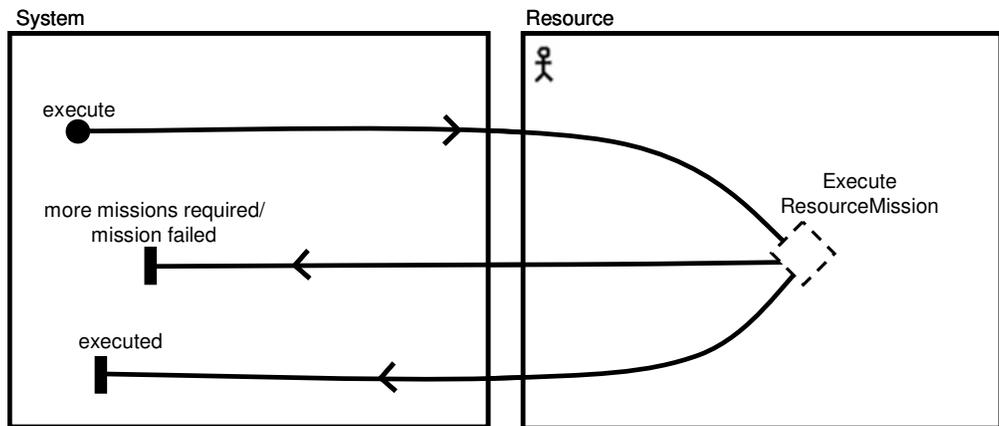


Figure 141 AoUCM: Resolve Crisis Use Case – ExecuteMission Plug-in Map

The responsibilities assignEmergencyLevel and setCrisisStatusActive are moved from the CaptureWitnessReport map to Figure 140, as these two responsibilities do not belong to the Communication concern to which the CaptureWitnessReport map is assigned. The

AoUCM model of the Resolve Crisis use case now focuses on the most fundamental elements of the crisis management system, i.e., the coordinator selecting missions, the missions being executed by resources either successfully or not, the mission file being updated accordingly, and the crisis file being closed when the crisis is resolved.

The remaining concerns described in this section extend the Resolve Crisis use case to define an AoUCM model that is equivalent to the UCM model from Figure 138 and Figure 139.

Recommend Strategies Concern

The Recommend Strategies concern is a rather straightforward concern that is required in two locations in the AoUCM model. Hence, the Recommend Strategies concern adds the system's recommendMission(s) responsibility before all responsibilities called selectMission(s) or indicateDesiredMission(s) (Figure 142). Note that selectMission(s) appears in the Resolve Crisis map in Figure 140 while indicateDesiredMission(s) appears on the Execute Super Observer Mission map [35].

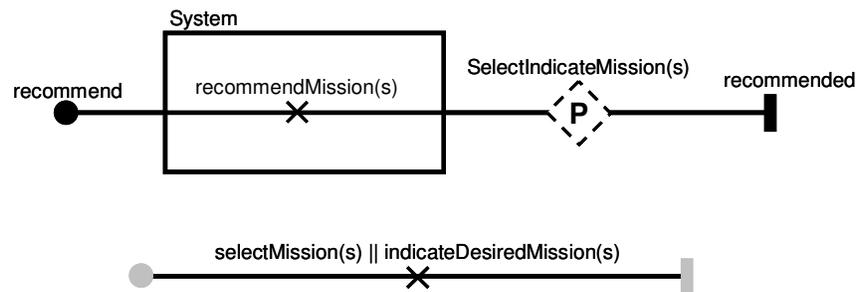


Figure 142 AoUCM: Recommend Strategies Concern

New Crisis and Mission Info Concern

The New Crisis and Mission Info sub-concern of the Communication concern highlights a more complex composition rule involving loops and concurrency in Figure 143. The sub-concern matches against the selectMission(s) responsibility of the Resolve Crisis use case. As visualized by the pointcut stub, an AND-fork is added after the selectMission(s) responsibility. This AND-fork spawns the handling of new information described earlier for Figure 138 while the base behavior continues. If changes are required, the spawned branch loops back to the pointcut stub (i.e., the coordinator selects missions again). Note that the newCrisisAndMissionInfoAvailable start point must be a local start point, since otherwise bindings between the aspect marker and the AoView of this concern cannot be

established automatically. Furthermore, the New Crisis and Mission Info sub-concern requires an aspect marker group.

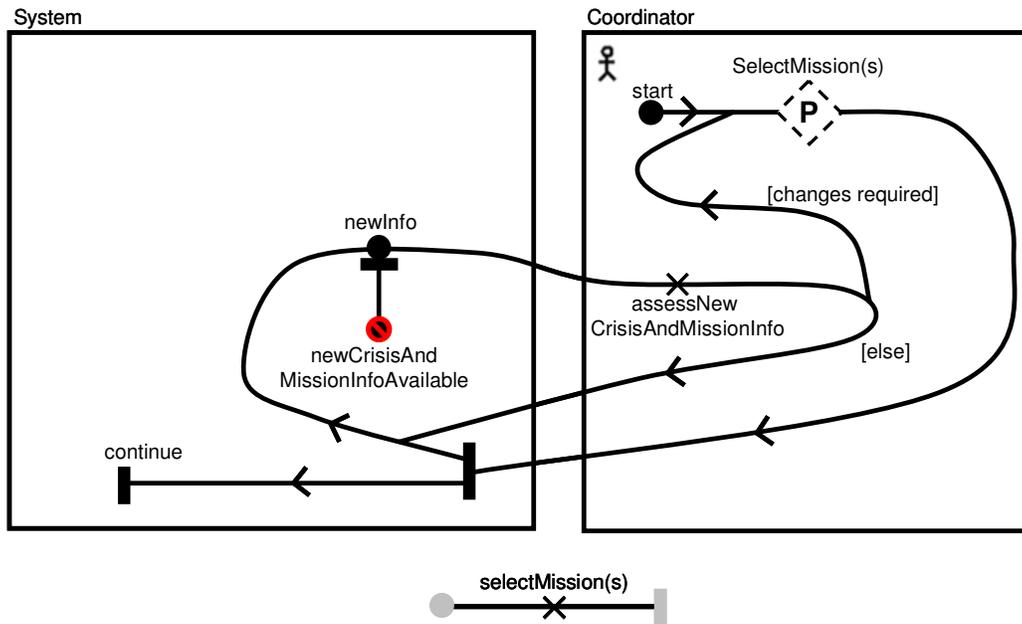
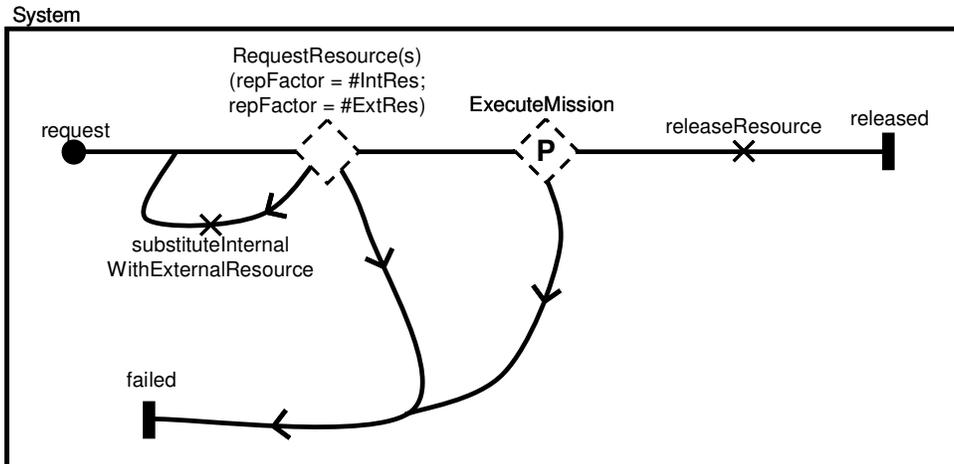


Figure 143 AoUCM: Communication Concern – New Crisis and Mission Info

Even though the pointcut expression of the sub-concern in Figure 143 identifies only one location and thus does not exhibit strong crosscutting, the sub-concern has to be viewed in the context of its parent concern Communication. The Communication concern cross-cuts other concerns several times as shown in Table 7. A complete version of Table 7 [35] shows all concerns affected by the Communication concern.

Resource Management Concern

The Resource Management concern requires a pointcut stub with more than one out-path (Figure 144), because the pattern that needs to be matched is the complete ExecuteMission map shown in Figure 141. Therefore, the ExecuteMission map is reused as is for the pointcut map of the Resource Management concern which is possible only because AoURN clearly separates aspectual properties from pointcut expressions. The start point of the ExecuteMission map is connected to the in-path of the ExecuteMission pointcut stub, the executed end point is connected to the out-path exiting the pointcut stub on the right, and the more missions required/mission failed end point is connected to the out-path exiting the stub at the bottom.



(Pointcut map not shown here since it is the same as Figure 141)

Figure 144 AoUCM: Resource Management Concern

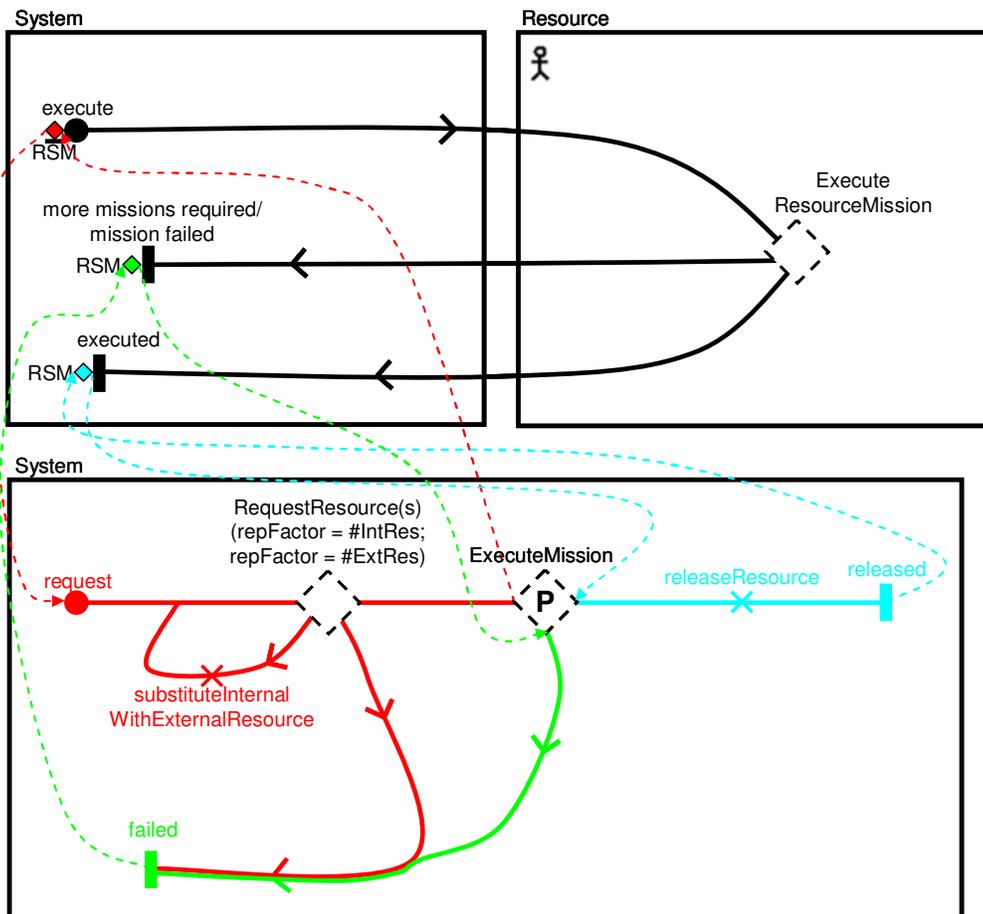


Figure 145 AoUCM: Resource Management Concern – AoViews

When the concern is applied, aspect markers are added before the start point and after the end points of the ExecuteMission map and bindings to the Resource Management concern are established (Figure 145). The aspect markers clearly visualize how much the Resource Management concern crosscuts the ExecuteMission map. The insertion of aspect markers results in the following effect. Before the execute start point is reached, the scenario follows the binding to continue on the concern map with requesting resources, possibly substituting internal resources with external resources. If the resource request is successful, the scenario returns to the ExecuteMission map by following the binding. Upon successful execution of the mission, the aspect marker after the executed end point is reached and the scenario follows the binding to continue on the concern map with releasing the resource. If, however, the resource request is not successful, the scenario ends with the failed end point on the concern map which is linked by a binding with the aspect marker after the more missions required/mission failed end point. The scenario therefore continues at this aspect marker, skipping the ExecuteResourceMission stub on the ExecuteMission map as desired. Consequently, the aspect markers of the Resource Management concern require an aspect marker group.

The model in Figure 145 highlights an omission in the CCCMS document: the handling of failed missions (e.g., when should resources be released). The CCCMS document is treated as a snapshot of the current understanding of the system, rather than a final, contractually-binding requirements document. In that sense, the models are built based on the document, reflecting the document as is with all its omissions with the understanding that the models would be reviewed again by the stakeholders. Care is taken, however, to ensure that the resulting models are consistent and major omissions are documented. Four major issues that should be addressed are a) the system currently assumes that only one witness report is captured per crisis, b) internal and external resources are treated differently, c) helicopter transport missions are mentioned only for transporting resources to the crisis location but not from the crisis location, and d) it is not specified in detail what happens when missions fail and the resource is already at the crisis location. Issue (a) possibly requires additional steps in Capture Witness Report to connect additional witness reports to an already existing crisis, and to possibly take no action if no new information is received. Issue (b) possibly requires external resources to

be treated more like internal resources instead of assuming that external resources will always respond to a request within an appropriate time frame. Issue (c) possibly requires another helicopter transport mission to be created after a resource finishes executing its mission and before confirming departure from the crisis location. Issue (d) possibly requires the behavior of each resource to be defined in more detail for the fail case.

Helicopter Transport Mission Concern

The Helicopter Transport Mission sub-concern (Figure 146) of the Coordination concern showcases a) the anything pointcut element and b) composition rules that can interleave two scenarios with each other. The purpose of this sub-concern is to check whether the resource must be transported to the crisis location by helicopter. The check must occur right after the execute start point of the ExecuteMission map. If the helicopter is not required, the scenario simply continues with the ExecuteMission map. If a helicopter is required, then the current mission is put on hold (wait for helicopter waiting place in Figure 146) and a helicopter mission is started. The start of the new helicopter mission, however, must be connected to the more missions required/mission failed end point of the ExecuteMission map in order to fit into the overall flow of the Resolve Crisis use case. Therefore, the Resolve Crisis use case and the Helicopter Transport Mission use case must be interleaved.

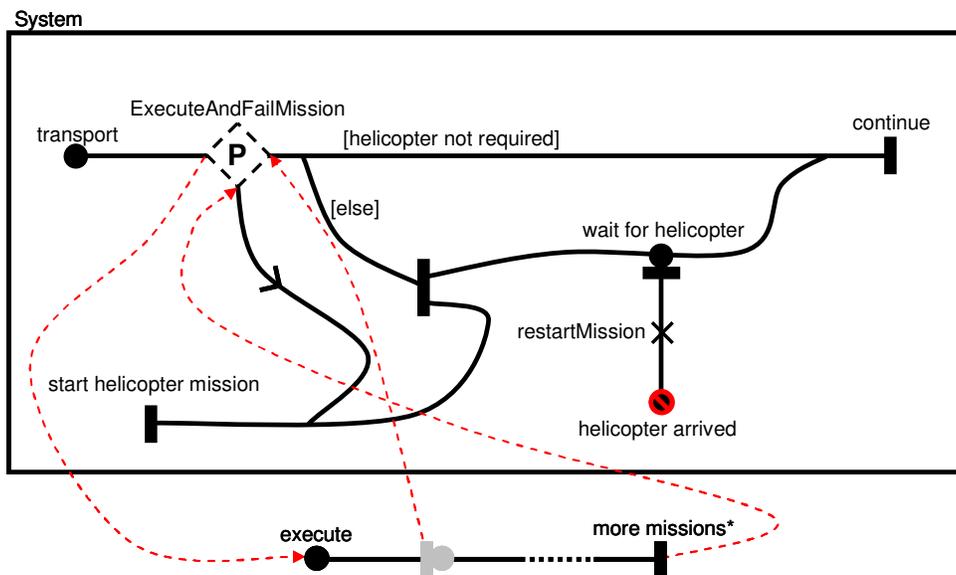


Figure 146 AoUCM: Coordination Concern – Helicopter Transport Mission

This interleaving composition is achieved by matching against the complete path from the execute start point to the more missions required/mission failed end point with anything in between as indicated by the anything pointcut element. The more missions* end point is then bound to the out-path exiting the pointcut stub at the bottom which connects to the start helicopter mission end point, ensuring that these two end points merge into one path. The end point of the connected end/start point pair is bound to the other out-path of the pointcut stub, specifying that whatever follows on this out-path will occur right after the execute start point and before anything matching against the anything pointcut element. Consequently, an aspect marker group is required for the Helicopter Transport Mission sub-concern.

The remaining concerns required to describe an AoUCM model that is equivalent to the UCM model from Figure 138 and Figure 139 are rather straightforward and, therefore, only briefly summarized here. The Capture Witness Report sub-concern of the Communication concern ensures that witness reports are captured before the Resolve Crisis use case starts. The Communicate with Resource at Location sub-concern of the Communication concern ensures that a resource confirms arrival at and departure from the crisis location, receives updates while at the crisis location, and submits a final report. Finally, the Super Observer Status Updates sub-concern of the Execute Super Observer Mission use case ensures that the super observer is informed of the creation, successful completion, or failure of a mission initiated by the super observer.

9.2.4 The Concern Interaction Graph (P5)

Several undesired interactions exist among the concerns of the CCCMS mentioned in Section 9.2.3, which are captured in the concern interaction graph in Figure 147. For example, the Recommend Strategies concern conflicts with the New Crisis and Mission Info concern and needs to be applied after it. The root of the conflict is that the pointcut maps of both concerns match against the `selectMission(s)` responsibility (Figure 142 and Figure 143). The Recommend Strategies concern adds the `recommendMission(s)` responsibility before the `selectMission(s)` responsibility (Figure 142), while the New Crisis and Mission Info concern adds (among other elements) an OR-join before the `selectMission(s)` responsibility (Figure 143). If the Recommend Strategies concern is applied first

and the New Crisis and Mission Info concern second, the OR-join will be closer to the selectMission(s) responsibility than the recommendMission(s) responsibility. In this case, the system will not recommend missions when new information becomes available because the OR-join merges into the path after the recommendMission(s) responsibility. Therefore, the resolution defines that the New Crisis and Mission Info concern must be applied before the Recommend Strategies concern.

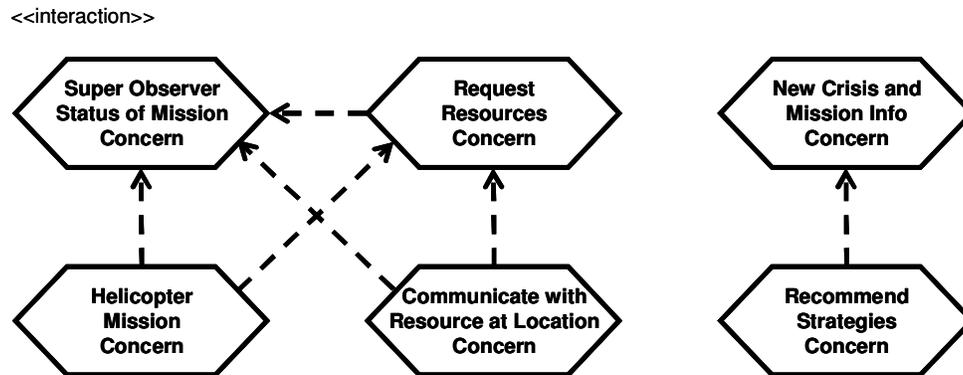


Figure 147 AoUCM: Concern Interaction Graph

Given the ordering of the concerns specified by the concern interaction graph, all concerns may now be applied to the AoUCM model. Figure 148 shows the complete AoUCM model for the Resolve Crisis use case with the automatically added aspect markers for all concerns mentioned in this section. Label CRL of an aspect marker stands for the Communicate with Resource at Location sub-concern, CWR for the Capture Witness Report sub-concern, HTM for the Helicopter Transport Mission sub-concern, NCM for the New Crisis and Mission Info sub-concern, RSM for the Resource Management concern, RST for the Recommend Strategies concern, and finally SOS for the Super Observer Status Updates sub-concern.

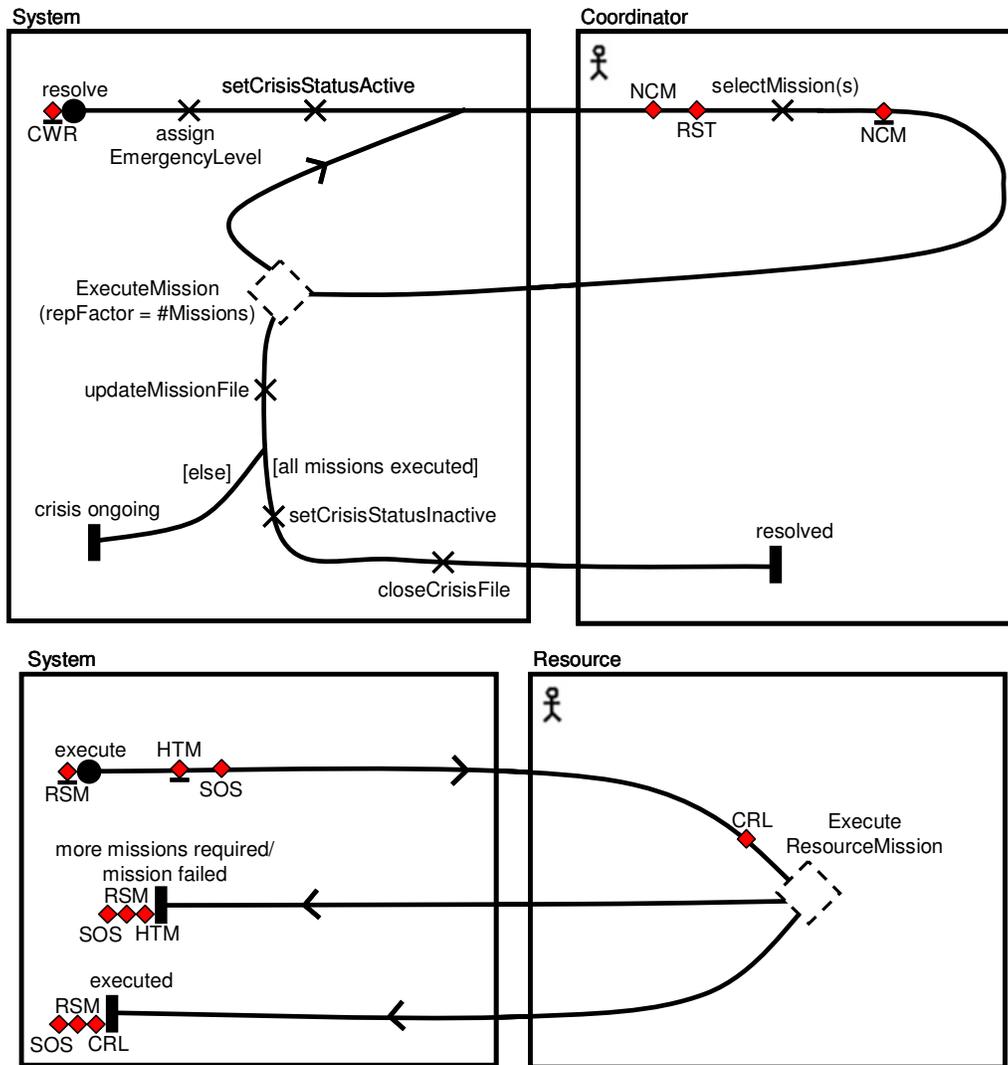


Figure 148 AoUCM: Resolve Crisis Use Case with Aspect Markers

9.2.5 Model Non-Functional Concerns with UCM and AoUCM (P3 – modified)

While the functional concerns discussed so far do crosscut each other to a certain extent, non-functional concerns tend to crosscut other concerns on a much larger scale. This is also true for the CCCMS challenge problem. NFR concerns can be thought of as layers that are added to the functional concerns, similar in fashion to placing transparencies with the layout of electrical wiring or plumbing on top of a floor plan of a house. The impact of NFR concerns is often widespread throughout the AoUCM model and often extremely time consuming and error-prone to model with standard UCM models due to the large

number of affected locations that have to be explicitly defined in the UCM models. To ensure consistency between the AoURN and URN models, the UCM models for the non-functional concerns are, therefore, created with the help of the AoUCM models which already identify the affected locations.

The UCM model for NFR concerns is derived by transforming the AoUCM model of the NFR concerns into a standard UCM model. Technically, this is achieved by a two-step transformation process. First, stubs are explicitly placed at each location where an aspect marker would be added by the AoUCM approach. The aspect maps are also transformed into standard maps by removing all pointcut stubs. New start points have to be added at the beginning of the pointcut stubs' out-paths and new end points at the end of the pointcut stubs' in-paths — creating several, possibly disjoint paths on the transformed aspect maps. Bindings are then established between the explicitly added stubs and the start and end points on the transformed aspect maps.

If an NFR concern makes use of variables on the aspect map, the aspect map is different for each aspect marker as the variable is replaced by the matched base elements. It is not pragmatic considering the effort required to create different plug-in maps in the UCM model — one for each stub that is added for an aspect marker — because some NFR concerns with variables add more than 50 aspect markers to the model. Instead, the variation is modeled with the help of component and responsibility plug-in bindings that ensure that the correct component and responsibilities are used. Therefore, a single plug-in map can still be used, avoiding a further explosion of model elements and maps in the UCM model.

Second, to further ensure the correctness and consistency of the AoUCM and UCM models, the UCM model is then carefully inspected for any locations that should not be affected by NFR concerns and for locations that are affected but are missed in the UCM models.

The transformation explained above ensures minimum impact on the maps in the UCM model that are affected by an NFR concern, while also ensuring maximum modularity/reusability as the maps for the NFR concerns are reused by several stubs. In addition, the size of the model remains minimal. It is thus a very appropriate candidate model to be compared with AoUCM. The resulting UCM model is consistent with the AoUCM

model in terms of content, but it is also a very efficient model that could be created by a modeler even without any aspect-oriented knowledge. The alternative approach is to explicitly add the behavior of an NFR concerns into the maps affected by the NFR concern. Again, this alternative is not feasible given that the NFR concerns Availability, Mobility, Persistence, and Security together are applied more than 250 times in the model.

The description of an NFR concern (e.g., in Section 2.3 of the CCCMS document) sometimes hints at further functionality that is not covered by the use cases later in the document. For example, managing resource and strategy information is mentioned in the Persistence NFR concern. In such cases, the NFR concern is applied only to the functionality described by the existing use cases as no new requirements are to be added to the challenge problem. However, the requirements for an NFR concern do introduce new functionalities for the concern itself (e.g., statistical data needs to be analyzed in the Statistical Logging NFR concern). AoUCM and UCM model these functionalities of NFR concerns. To be more precise, AoUCM and UCM model the operationalization of NFR concerns, i.e., a particular behavior that ensures the desired characteristics of the NFR concern. The description may be as detailed or abstract as required, reflecting how much is known about the NFR concern at the time the model is built. As the CCCMS document does not address the behavioral aspect of NFR concerns at all, the AoUCM and UCM description of the NFR concerns is kept as abstract as possible to minimize the introduction of new requirements.

Availability NFR Concern

The Availability NFR concern in Figure 149 describes, in a very abstract way, redundancy as a means to achieve availability. Various means of achieving availability may be described in the goal model of the concern, hence capturing the reasons for choosing one solution for availability over another. Each solution may then be described in more detail in the scenario model. For the redundancy solution to availability, each responsibility of the system is shadowed in parallel by the backup system. The Availability NFR concern makes use of the variable \$Task. As the pointcut map matches against any system responsibility, this is clearly an NFR concern with a very significant impact on the AoUCM model.

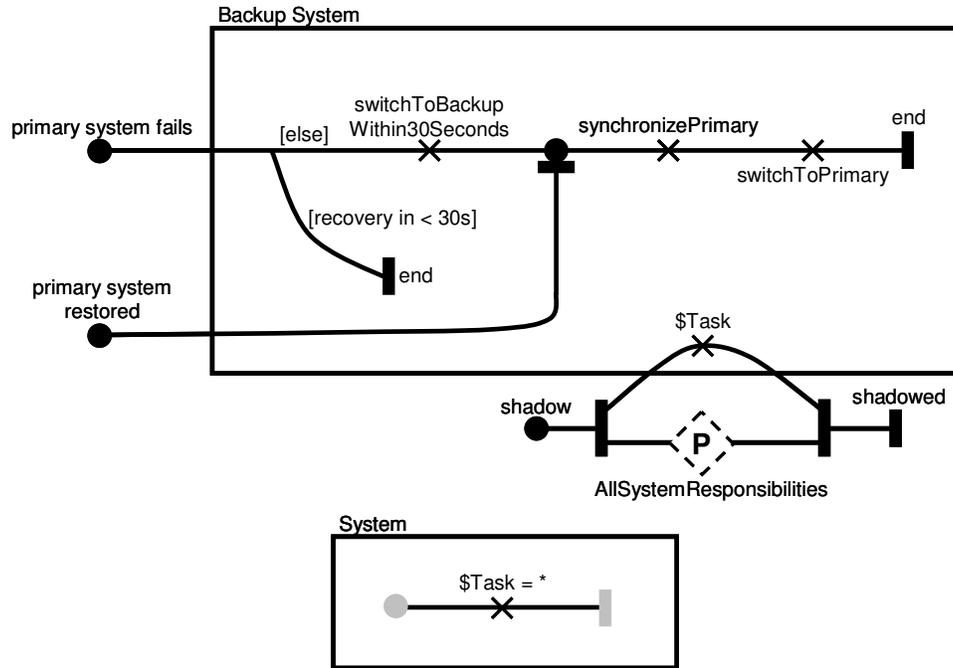


Figure 149 AoUCM: Availability Concern

In addition to the path with the pointcut stub, the aspect map of the Availability NFR concern also defines with a separate path what happens if the primary system fails. This separate path defines concern-specific behavior described at a very high level of abstraction. It could also be described on a separate map without changing the meaning of the AoUCM model.

Mobility NFR Concern – Infrastructure

The Infrastructure sub-concern of the Mobility NFR concern in Figure 150 is another example of a concern with wide-reaching impact. It defines two patterns with the help of two pointcut maps that are both plugged into the pointcut stub on the aspect map. The first pattern captures all interactions between the system and a number of mobile actors where the system is the sender. The second pattern captures a similar situation but with the system as the receiver. The NFR concern then replaces the matched pattern (i.e., the sender and receiver components) with the behavior specified on the aspect (i.e., the sender and receiver are added back again but this time with the Wireless Infrastructure functionality in the middle).

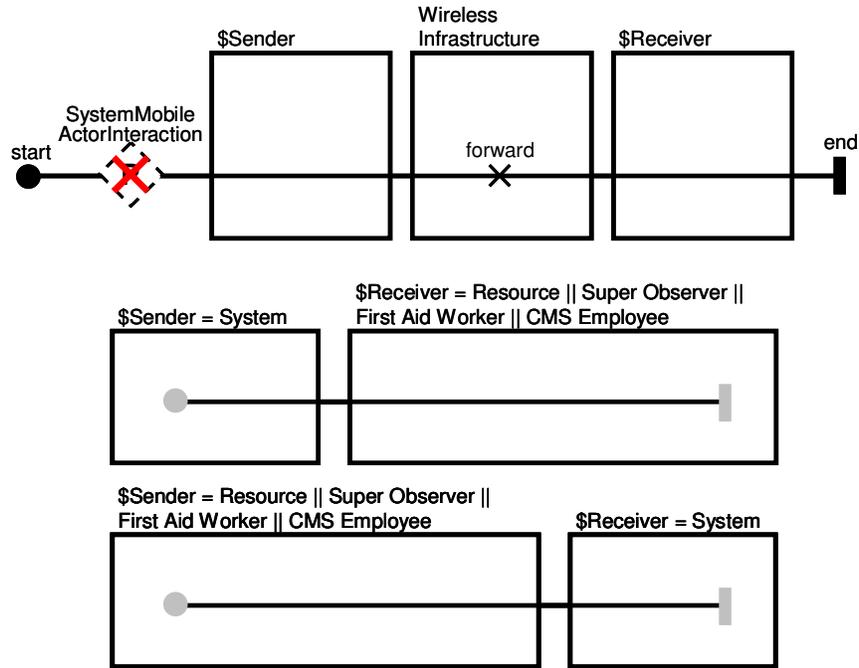


Figure 150 AoUCM: Mobility Concern – Infrastructure

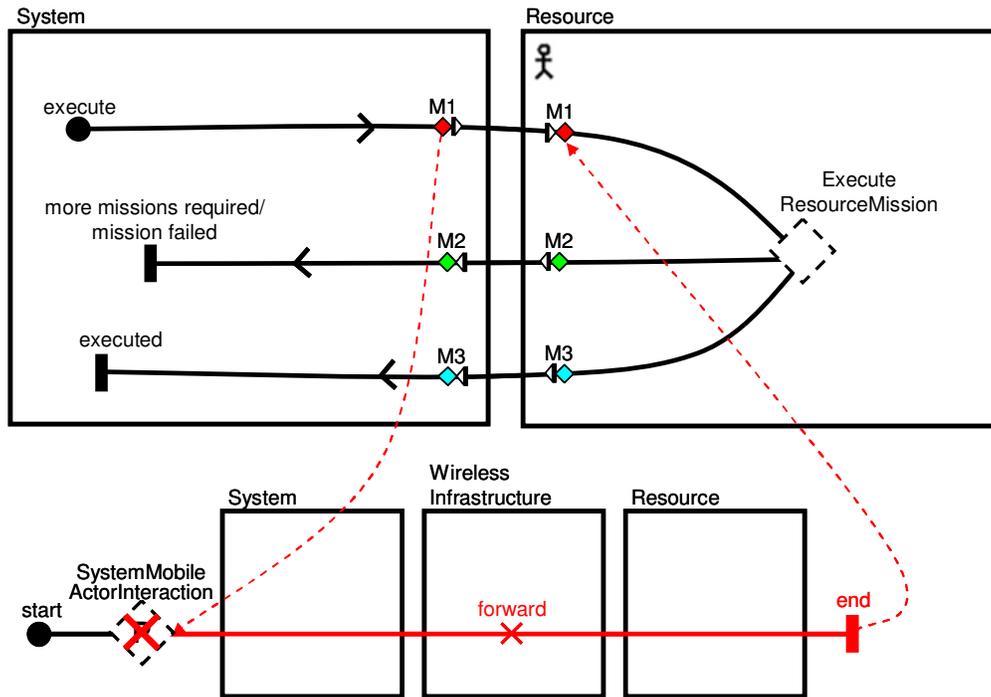


Figure 151 AoUCM: Mobility Concern – Infrastructure – AoView

The pointcut expressions of the Infrastructure concern are matched three times on the ExecuteMission map shown in Figure 151 as indicated by the three pairs of tunnel en-

trance and exit aspect markers. For the top pair, the System and Resource are matched against \$Sender and \$Receiver, respectively, by the first pointcut map in Figure 150. The AoView at the bottom of Figure 151 clearly indicates this by replacing \$Sender with System and \$Receiver with Resource in the aspect map. For the other two pairs of aspect markers, Resource is matched against \$Sender and System is matched against \$Receiver.

Safety NFR Concern – Weather Information System

The last NFR concern to be discussed for AoUCM in this section is the Weather Information System sub-concern of the Safety NFR concern in Figure 152. Its purpose is to monitor the weather and determine if a safety perimeter is required for the crisis location. This monitoring is an ongoing activity throughout the crisis. Therefore, the monitoring starts at the time the crisis status is set to active. It ends when the crisis is resolved. Similar to the Helicopter Transport Mission concern from Figure 146, a path from the setCrisisStatusActive responsibility to the resolved end point is matched with the help of the anything pointcut element. An AND-fork before the pointcut stub on the aspect map spawns the parallel behavior of the NFR concern (i.e., just before the setCrisisStatusActive responsibility at the beginning of a crisis). A second AND-fork provides new weather information on one branch and on the second branch sets a timer which ensures that the weather is checked periodically. When a timeout occurs, the scenario loops back, thus providing more weather information and setting the timer again. This loop continues until the timer is deactivated by the out-path of the pointcut stub which is connected to the resolved end point (i.e., it is deactivated after the resolved end point is reached at the end of the crisis).

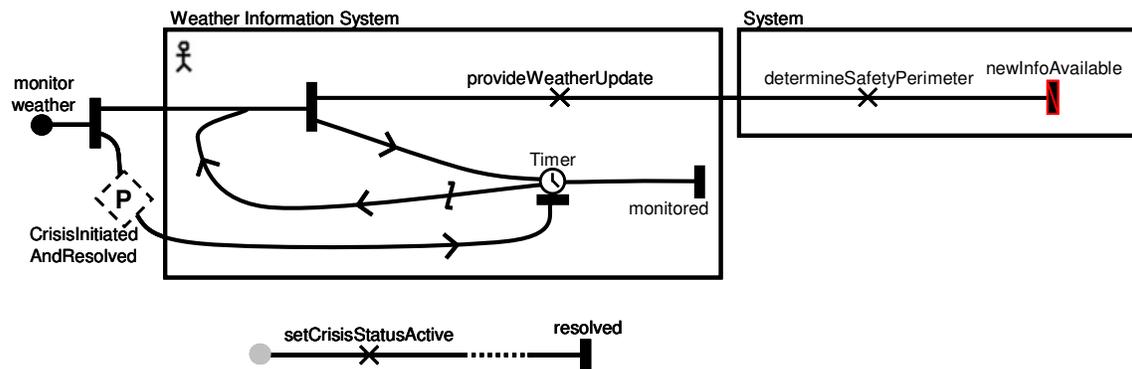


Figure 152 AoUCM: Safety Concern – Weather Information System

9.2.6 Model Stakeholder Concerns with GRL and AoGRL (P4 – modified)

The objectives of the CCCMS and the high-level goals associated with the NFRs in the CCCMS document are first modeled in GRL and AoGRL models that focus on the dependencies between stakeholders. The result of this analysis is shown for GRL in Figure 153. In addition to the dependencies, Figure 153 shows the goal model of the Coordinator stakeholder in more detail. In general, the Government, the Resource, and the General Public depend on the Coordinator for improving customer QoS metrics, for mission requests that allow a resource to handle an incident efficiently, and for receiving appropriate help in a timely fashion, respectively. The General Public also depends on the Resource to do its job, while the Coordinator depends on the Resource to provide updates about its status and location and on the General Public to report the incident in the first place.

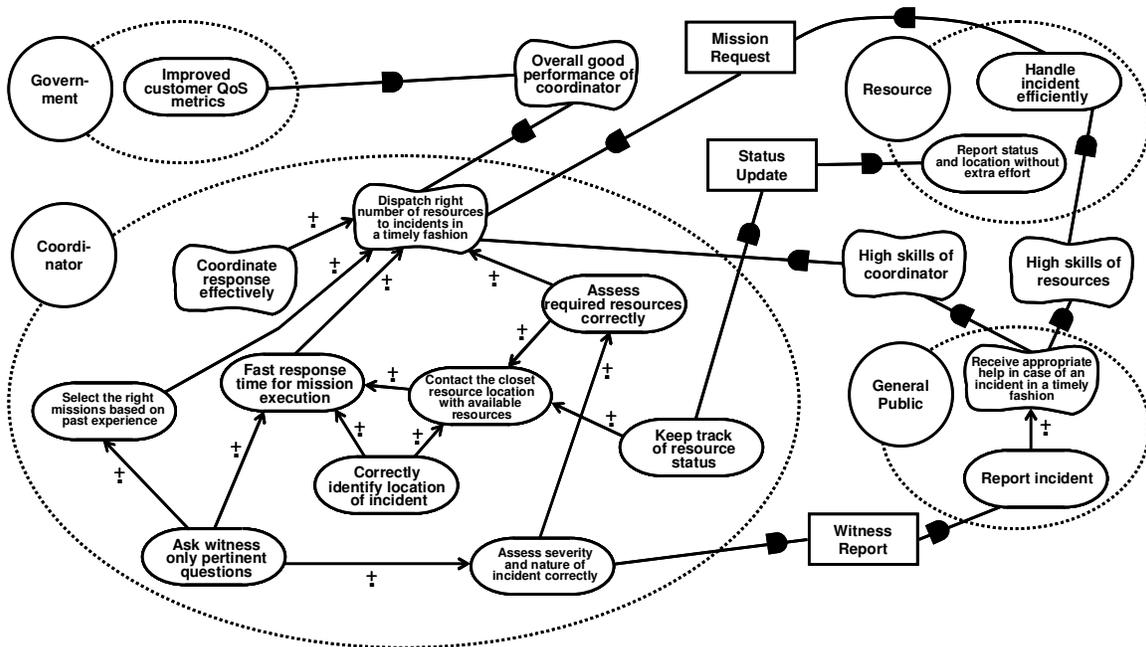


Figure 153 GRL: Stakeholder Dependencies

The AoGRL version of the GRL model in Figure 153 simply separates each stakeholder into its own goal graph, repeating each dependency as required. This is a case where aspect-oriented thinking provides guidance in how to structure the views of the goal model, even though aspect-oriented techniques are not required to achieve this. Standard separation of concerns with basic GRL modeling is sufficient.

The GRL goal model in Figure 154 illustrates the impact of all functional concerns (i.e., use cases) on the stakeholder goal graphs. Typically, tasks model solutions that impact positively or negatively some of the goals specified for one or more stakeholders. The solutions may then be described in more detail with UCM models as is the case here (e.g., the general Resolve Crisis task corresponds to the detailed Resolve Crisis use case map).

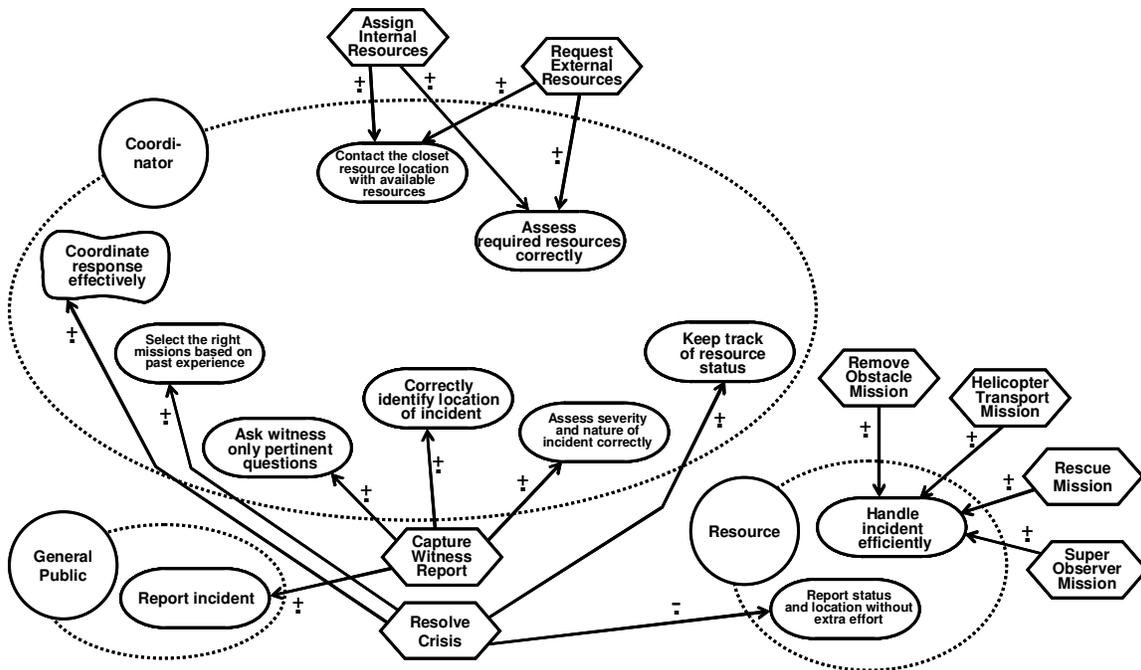


Figure 154 GRL: Impact of Use Cases on Stakeholders

Similar to before, the AoGRL model splits the single goal graph into individual AoGRL goal graphs — one for each functional concern. However, this split is not a one-to-one transformation, because the tangling that exists in the UCM model is carried over into the GRL model. For example, the Resolve Crisis task has an impact on the Select the right missions based on past experience goal only because the recommendMission(s) responsibility is part of the Resolve Crisis use case map in the UCM model. In the AoUCM model, however, this is factored out into the Recommend Strategies concern. Therefore, the impact on the Select the right missions based on past experience goal is shown on the Recommend Strategies AoGRL goal graph but not on the Resolve Crisis AoGRL goal graph.

Finally, the NFR concerns are also modeled with GRL and AoGRL. As an example, the Availability NFR concern is shown in Figure 155 for GRL and in Figure 156 for AoGRL. The two models are equivalent in terms of content but are structured very differently. In both cases, the availability requirement 2hrs of downtime every 30 days, failure recovery within 30sec is first identified in the CCCMS document. Then, it is decided which goals of which stakeholders are affected by this requirement. Figure 155 shows that the Government and the General Public have an interest in this requirement and therefore the Availability NFR concern. Since the Government and the General Public depend on the Coordinator and the Resource (Figure 153), the availability requirements also depend on the Availability softgoals of the Coordinator and the Resource. This relationship is modeled by the dependencies between the availability requirement and the Availability softgoals. Last but not least, one has to consider not only the stakeholders but also the functional concerns. Any functional concern that has an impact on the Coordinator or the Resource now also depends on the Availability softgoal. In other words, a functional concern cannot be considered achieved, if availability for the functional concern is not provided for the stakeholder. Therefore, dependency links are established between the tasks and the Availability softgoals.

The AoGRL version of the Availability NFR concern rather describes the reasons behind the dependencies that are established explicitly in Figure 155 by the standard GRL approach. The pattern in Figure 156 states that dependencies (the ones without markers) should be added to the model as long as there exists a dependency from a stakeholder interested in the availability requirement (shown on the right of Figure 156) to another stakeholder (shown on the left in Figure 156). In this case, the Availability softgoal is added to the stakeholder on the left and dependencies are established to the stakeholder on the right and any tasks that impact the stakeholder on the left.

The top part of the pointcut expression in Figure 156 therefore matches against the Government stakeholder with the Improved customer QoS metrics goal and the General Public stakeholder with the Receive appropriate help in case of an incident in a timely fashion softgoal, because dependencies exist from these two intentional elements to other stakeholders (i.e., to Coordinator and Resource). Hence, the 2hrs of downtime every 30 days, failure recovery within 30sec softgoal is added to the Government and

General Public stakeholders, the Availability softgoal is added to Coordinator and Resource stakeholders, and dependencies are added between these two softgoals.

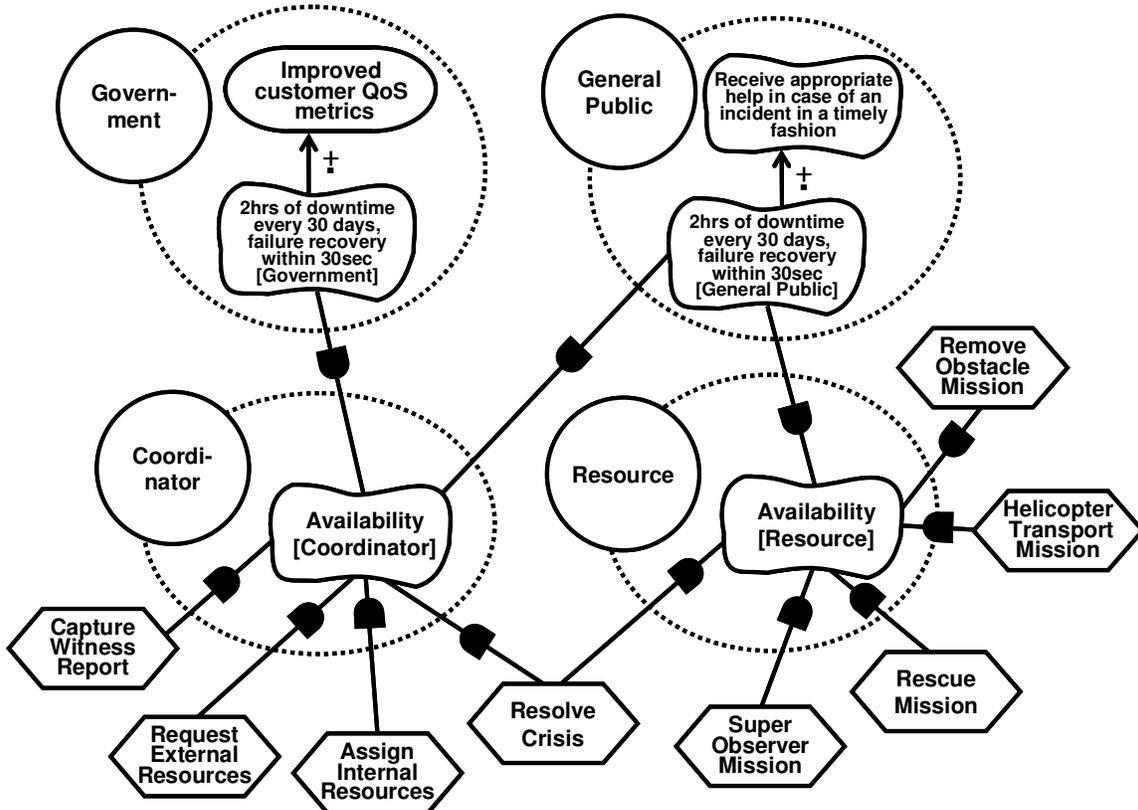


Figure 155 GRL: Availability NFR Concern

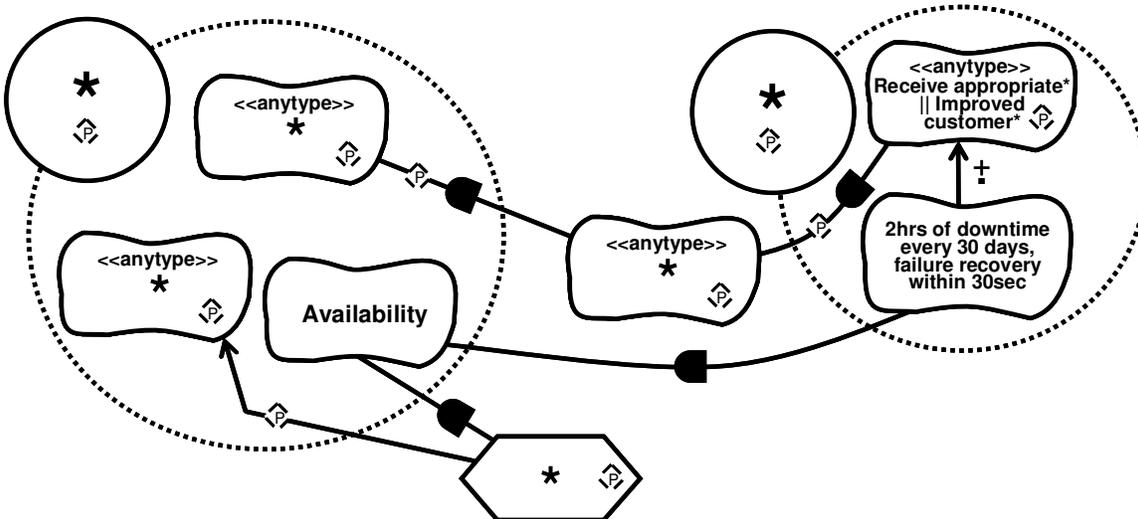


Figure 156 AoGRL: Availability NFR Concern

Furthermore, all tasks contributing to an intentional element in the Coordinator and Resource stakeholders now also depend on the Availability softgoal (e.g., the Capture Witness Report task and the Super Observer Mission task). While the pointcut expression shown in Figure 156 is very general as it uses several pointcut elements with a *, it may be further restricted as needed by replacing the * with more specific expressions. For example, the pointcut expression could be restricted to specific tasks or to specific intentional elements in the left actor in Figure 156 or to a specific dependum. If required, the Availability goal of the concern can be further refined on one or more separate goal graphs of the concern, showing alternative, ideally generic solutions for achieving availability. For example, such as separate goal graph exists for the redundancy solution described in the AoUCM model [35].

9.3. YKeyK System

The Your Key Knows (YKeyK) system allows drivers to find their car in a car park by following directions shown on a small display of the car key [2]. It has been used as a case study on several occasions [2][4][122]. The complete system model consists of six stakeholders, five NFRs, and three use cases. The NFRs are reasoned about in the goal models but not further refined with scenario models. Three models of the YKeyK system are created – two URN models (YKeyK.A, YKeyK.B) and one AoURN model (YKeyK.C).

1. **YKeyK.A** – This is a URN model that contains only one goal graph and one map with the complete system. The GRL model is similar to Figure 105.a on Page 155, but without the references to Figure 105.b and with instances of the referenced goal graph directly copied into Figure 105.a instead. The UCM model is a merge of Figure 107, Figure 108, and Figure 109 where the user first enters the car park, then searches for the car, and then exits the car park. The aspectual behavior is also merged into the map. With a total of almost 300 model elements in one goal graph and more than 60 model elements in one map, the approach for YKeyK.A is clearly not scalable and mostly serves as a base to compare other alternatives.

2. **YKeyK.B** – This is the most likely used approach to describe the YKeyK system with URN. Each stakeholder is described on its own goal graph, referencing if necessary goal graphs for non-functional requirements and use cases. This model is similar to Figure 105, but splits up Figure 105.a into separate goal graphs, one for each stakeholder. Only outgoing dependency links are shown on a stakeholder goal graph in order to avoid redundant incoming dependency links (as they are covered by outgoing links on another stakeholder goal graph). In addition to the shown goal graph, goal graphs for use cases simply specify the links between the tasks related to a specific use cases (e.g., the tasks prefixed with UC002 in Figure 105.a). This approach assumes that parameterized softgoals are supported by GRL (e.g., the Performance softgoals with [...] in Figure 105.a). The UCM model is similar to Figure 107, Figure 108, and Figure 109 but with the aspectual behavior directly included on each of the three maps.
3. **YKeyK.C** – This is an AoURN model of the YKeyK system as shown in Figure 106 to Figure 109. Similar to YKeyK.B, each stakeholder, each non-functional requirement, and each use case is encapsulated by a concern and modeled on separate goal graphs. The stakeholder goal graphs, however, do not reference the goal graphs for non-functional requirements and use cases. Instead, pointcut graphs for each non-functional requirement and use case define how the crosscutting non-functional requirements or use cases are added to the model. Similarly, the AoUCM model describes aspectual behavior separately.

9.4. Quantitative Metrics

This section discusses the metrics used to assess the AoURN and URN models in the model-based evaluation as well as the common modeling tasks performed on these models in the task-based evaluation.

Model-based Evaluation

The evaluation of the AoURN and URN models is carried out based on metrics for modularity, reusability, and maintainability of aspect-oriented software available in the literature [133]. The metrics are adapted for the use in URN and AoURN models by mapping the notion of component, operation, and line of code in [133] to URN and AoURN model elements as explained in Table 8. As the metrics are merely adapted, the quality model still applies to this evaluation. The quality model links the metrics to the qualities of interest such as reusability and maintainability and is established in [133] with the help of Basili's Goal-Question-Metric (GQM) methodology [26]. Furthermore, metrics for separation of concerns are directly related to modularity.

Table 8 Mapping for Adapted Metrics

Notion from [133]	URN/AoURN Model Elements	
Component	Model Unit (MU)	goal graph, use case map
Operation	Key Model Element (KME)	intentional element, responsibility, stub (including references)
Lines of Code (LOC)	Model Element (ME)	any visual URN/AoURN model element (including references and concerns) ¹⁾

¹⁾ Direction arrows and empty points are not counted.

While the metrics in [133] were established to assess aspect-oriented implementations at the programming level, they do also apply to the modeling level as the difference is mainly a matter of abstraction. The concepts used for the original metrics are components, operations, and lines of code (LOC) and hence rather programming-centric. However, these programming-centric concepts relate to much larger concepts generally applicable to all modeling levels: containers, behavioral units, and elementary units. The mapping from the original programming-centric concepts to AoURN and URN concepts retains these generally-applicable concepts.

Components are mapped to *model units*, i.e., goal graphs and use case maps, as they are containers of goals trees and scenario paths. Operations are mapped to *key model elements*, i.e., the behavioral units of GRL and UCM models described by intentional elements and responsibilities/stubs, respectively. Lines of code are mapped to *model elements*, as any modeling element may be considered an elementary unit. Therefore, the general structure that is assessed by the metrics from [133] at the programming level also

exists in AoURN and URN models, allowing us to apply the metrics at a different abstraction level.

The definition of each metric is summarized in Table 9. In general, the lower the result of a metric, the better it is. The metrics in group A measure the identified concerns, the metrics in group B measure the model units (i.e., goal graphs and use case maps), while the metrics in group C measure the whole model.

Table 9 Definition of Metrics

<i>A) Separation of concerns metrics [Measures concerns]</i>
CDMU (Concern Diffusion over Model Units) Count the number of MUs whose main purpose is to contribute to a concern plus all other MUs that reference them.
CDKME (Concern Diffusion over Key Model Elements) Count the number of KMEs whose main purpose is to contribute to a concern .
CDME (Concern Diffusion over Model Elements) Count the number of concern switches from a concern to other concerns in each MU.
<i>B) Coupling and cohesion metrics [Measures model units]</i>
CBMU (Coupling between Model Units) Count the number of other MUs on which an MU depends.
LCOKME (Lack of Cohesion in Key Model Elements) Count the number of KMEs in an MU that do not contribute to the MU's main concern.
<i>C) Size metrics [Measures the model]</i>
VS (Vocabulary Size) Count the number of MUs and concerns in the model .
NME (Number of Model Elements) Count the number of MEs in the model .
AvNME (Average Number of Model Elements) Calculate the average number of MEs per goal graph and map in the model .

Task-based Evaluation

In addition to the metrics, the most common update tasks are considered for AoURN and URN models. These update tasks cover adding, removing, or changing a non-functional requirement, use case, or stakeholder. Each use case, NFR, and stakeholder defined in the URN model and AoURN model is considered individually and a worst-case analysis is performed, i.e., those model units (use case maps and goal graphs) in the model are counted that may have to be updated due to changes in the requirements document. Finally, the counts are averaged over the number of use cases, NFRs, and stakeholders to be able to compare all three categories and the models with each other. The lower the result, the better it is.

9.5. Comparison of URN and AoURN Models

9.5.1 Car Crash Crisis Management System

The comparison of URN and AoURN models of the CCCMS is based on the models described in Section 9.2. The comparison of GRL and AoGRL is based on the models described in Section 9.2.6. For the AoUCM model, the models described in Sections 9.2.3 and 9.2.5 are used. For the UCM model, the models described in Section 9.2.2 and 9.2.5 are used.

Model-based Evaluation

As an example for the metrics calculation and illustration of the advantages of AoUCM over UCM, the Resolve Crisis UCM model is presented in Figure 157. The corresponding AoUCM model is shown in Figure 140 on Page 236. The stubs in the Resolve Crisis UCM model address all modeled NFR concerns. Label AV of a stub stands for Availability, PE for Persistence, CAM for Criminal Activity Monitoring System, EM for Emissions Monitoring System, WI for Weather Information System, AU for Authentication, EE for Encryption (Encrypt), ED for Encryption (Decrypt), and finally ST for Statistical Logging.

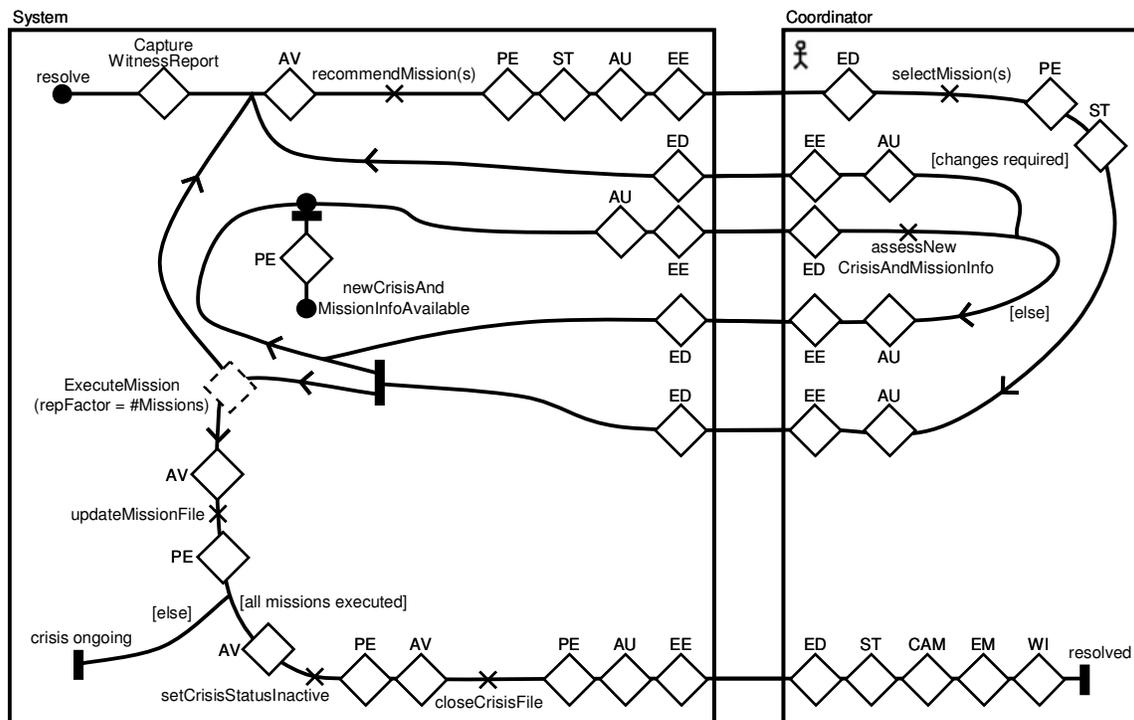


Figure 157 UCM Model for the Resolve Crisis Use Case with All NFR Concerns

For the CDMU metric, the UCM model in Figure 157 counts towards the Resolve Crisis concern and also towards each of the eight NFR concerns represented by stubs, because the stubs reference the main maps of these concerns. The AoUCM model in Figure 140 counts only towards the Resolve Crisis concern. Therefore, the score is 1 for AoUCM and 9 for UCM.

For the CDKME metric, the UCM model must count all 34 stubs that are added because of the NFR concerns and the Capture Witness Report stub. In addition, all other N responsibilities and stubs shown in Figure 140 and Figure 157 also have to be counted. However, the counts of these model elements even out since all these model elements are used on some map in both complete models. Therefore, the score is $0+N$ for AoUCM and $35+N$ for UCM.

For the CDME metric, 75 concern switches exist in the UCM model while no concern switch exists for the AoUCM model. The concern switches are calculated for each concern on each map. For example, there are two concern switches for the Recommend Strategies concern – one from the AV stub to recommendMission(s) and then one from recommendMission(s) to the PE stub. Therefore, the score is 0 for AoUCM and 75 for UCM.

For the CBMU metric, all plug-in maps have to be counted. For UCM, there is one in CaptureWitnessReport, two in ExecuteMission, plus the eight NFR concerns. For AoUCM, there are the two in ExecuteMission. Usually, pointcut maps increase the count for AoUCM models, but the Resolve Crisis concern does not have any in this case. Therefore, the score is 2 for AoUCM and 11 for UCM.

For the LCOKME metric, all 34 stubs of the NFR concerns, the CaptureWitnessReport stub, the recommendMission(s) responsibility, and the assessNewCrisisAndMissionInfo responsibility must be counted. For AoUCM, the count is zero. Therefore, the score is 0 for AoUCM and 37 for UCM.

For the VS metric, all concerns as well as the pointcut maps must be counted for the AoUCM model but do not have to be counted for the UCM model since they do not exist in the latter model. The use case map must be counted in both cases. Therefore, the score is 2 for AoUCM and 1 for UCM. Note that the Resolve Crisis concern does not

have a pointcut map. The other concerns in the complete AoUCM model, however, have at least one and as many as four pointcut maps.

For the NME metric, the concerns have to be counted for the AoUCM model while the stubs added by the NFR concerns and the Capture Witness Report concern have to be counted for the UCM model. In addition, all other N model elements shown in Figure 140 and Figure 157 including the maps themselves also have to be counted. However, the counts of these model elements even out since all these model elements are used on some map in both complete models. Therefore, the score is $1+N$ for AoUCM and $35+N$ for UCM.

Finally, the AvNME metric divides the NME metric by the number of goal graphs and use case maps in the system. It is therefore meaningless to compute this metric for only one use case map.

The assessment of the GRL and AoGRL models of the CCCMS is done in an analogous way to the above example. The overall results of the metrics assessment are shown in Table 10 and Table 11 with the better result shown with a grey background. The coupling and cohesion metrics (CBMU, LCOKME) are significantly better for AoURN than for URN as are the separation of concern metrics (CDMU, CDKME, CDME). In short, AoURN outperforms URN for all metrics except the Vocabulary Size (VS).

The AoURN-based approach introduces new modeling elements — namely concerns, pointcut maps, and pointcut graphs — into the AoURN model that are not needed for the URN model, explaining the larger VS metric for the AoURN model. The concerns, however, are used to group the AoURN model into more manageable sub-models that each contain a very small number of diagrams. Furthermore, these diagrams are simpler as is shown by the better results of AoURN for the Number of Model Elements (NME) metric and the Average Number of Model Elements (AvNME) metric. Essentially, the AoURN model is generally smaller, with less complex diagrams than the URN model. AvNME is reduced by a factor of three for AoURN models compared to URN models. Moreover, concerns are encapsulated in AoURN models to a greater extent than in URN models as demonstrated by the separation of concerns, coupling, and cohesion metrics. Despite the better results of VS for URN, it cannot be concluded that URN is a

better approach than AoURN. There is a trade-off between the complexity of model units and the number of concepts and modeling elements in the model.

Table 10 CCCMS Results of the Separation of Concerns Metrics

	<i>UCM</i>			<i>AoUCM</i>		
	CDMU	CDKME	CDME	CDMU	CDKME	CDME
Functional Concerns	25	123	179	41	125	29
NFR Concerns	112	316	691	31	58	9
Stakeholder Concerns	n/a	n/a	n/a	n/a	n/a	n/a
Total	137	439	870	72	183	38

	<i>GRL</i>			<i>AoGRL</i>		
	CDMU	CDKME	CDME	CDMU	CDKME	CDME
Functional Concerns	0	28	20	21	20	10
NFR Concerns	11	46	6	6	40	0
Stakeholder Concerns	3	58	13	4	45	11
Total	14	132	39	31	105	21

Grand Total	151	571	909	103	288	59
--------------------	-----	-----	-----	-----	-----	----

Table 11 CCCMS Results of the Coupling, Cohesion, and Size Metrics

	CBMU	LCOKME	VS	NME	AvNME
UCM	110	378	29	749	26
GRL	8	34	7	187	27
Total – URN	118	412	36	936	27
AoUCM	45	6	90	594	9
AoGRL	13	14	39	206	11
Total – AoURN	58	20	129	800	9

AoURN clearly outperforms URN for NFR concerns in terms of the separation of concerns metrics. Interestingly, when focusing on the functional concerns alone (Table 10), the Separation of Concern metrics CDMU and CDKME are slightly better for UCM models than for AoUCM models. The difference for the CDKME metric is insignificant. The results of the CDMU metrics can be explained by the fact that the CDMU metric for a model with only one model unit (i.e., only one huge goal graph or one huge use case map) has the lowest possible score. It has to be considered in conjunction with the CDME metric, which measures tangling. Since the CDME metric is significantly higher for URN

models than for AoURN models, it is suggested that AoURN still performs better than URN.

The same reasoning applies to the CDMU metrics that are better for GRL compared to AoGRL and the slightly lower CBMU value for GRL compared to AoGRL. AoGRL splits the single GRL graph which describes the impact of the functional concerns on the stakeholder goals into individual graphs — one for each functional concern. This split introduces coupling to the stakeholder concerns from many functional concern graphs. However, a single goal graph for capturing this impact is not feasible for larger systems as demonstrated again by the CDME metric.

The slightly lower count for NME can also be explained by the single GRL goal graph being split into many for the AoGRL model. The split introduces many duplicated modeling elements. However, the resulting goal graphs are much smaller as evidenced by the AvNME metric. Again, there is a trade-off between the complexity of individual model units and the overall number of modeling elements.

Task-based Evaluation

As examples for the update tasks investigated by the task-based evaluation, consider the omissions stated towards the end of Section 9.2.3. Note that the set of use cases is slightly different for URN and AoURN models, because the aspect-oriented analysis of the use cases in the CCCMS document resulted in a restructuring of the use cases. Therefore, updates to the original use cases are investigated for the URN model, while updates to the restructured use cases are investigated for the AoURN model. In contrast to the use cases, NFRs and stakeholders remain the same for both models. There is a section for each NFR in the non-aspect-oriented CCCMS document and a concern for each NFR in the AoURN model. The URN model reflects the CCCMS document. Similarly, each stakeholder is described in the non-aspect-oriented CCCMS document and a concern exists for each stakeholder in the AoURN model.

Table 12 shows the results of the task-based evaluation with the better result shown with a grey background. In general, the difference for updates to use cases is insignificant. Furthermore, there is a difference for updates to stakeholders which can be explained by the huge goal graphs in the URN model. The URN model contains one huge goal graph for all stakeholders and another one for all use cases, while the AoURN model

breaks these two goal graphs into several individual ones for each stakeholder and each use case. Consequently, an update of a stakeholder in the AoURN model may affect more goal graphs than an update in the URN model. This is a trade-off against huge goal graphs that are difficult to maintain and very quickly become unwieldy.

Table 12 CCCMS Results of the Task-Based Evaluation

	<i>Average Number of Diagrams Needing Modifications per Update Task</i>			
	Use Cases	NFRs	Stakeholders	Total
URN	4.11	9.83	3.50	6.76
AoURN	3.85	3.33	6.00	3.93

Finally, there is a significant difference for updates of NFRs. The results of the task-based evaluation nicely indicate the amount of crosscutting that occurs for use cases and NFRs. The results for use cases are not very different because only some crosscutting occurs for use cases. In this situation, the traditional URN model and the AoURN model, with its strict encapsulation of anything related to a particular use case or feature, perform similarly. NFRs, however, crosscut other concerns very broadly and are therefore prime candidates for aspect-oriented modeling. The results of the task-based evaluation reflect this.

When updating an NFR in the URN model, the goal graphs and use case maps for the NFR require updating. Then, all locations requiring changes in the stakeholder goal graphs as well as the goal graphs and use case maps of other use cases and NFRs must be found and also changed. For use cases and stakeholders, the updates are more localized than for NFRs, but generally the changes are distributed over many different goal graphs and use case maps. For AoURN models, on the other hand, the update tasks require changes to the aspect graphs, aspect maps, pointcut graphs, and pointcut maps of one single concern (either the one encapsulating the use case, the NFR, or the stakeholder). Even though these updates may affect several use case maps and goal graphs, they are all generally localized to one group of use case maps and goal graphs contained in one concern.

There is, however, one notable exception. Changes to the AoURN model may impact the matches of the pointcut expressions defined for concerns in the AoURN model. Hence, the pointcut expressions may also have to be changed. With tool support, the changes to the matches may be monitored and reviewed, ensuring that the desired lo-

cations in the AoURN model are stilled matched correctly. It is difficult to quantify how important the updates required for pointcut models due to changes in requirements are. Good tool support is certainly required. The trade-off here is between changing crosscutting concerns in non-aspect-oriented models and keeping pointcut expressions up-to-date when aspect-oriented models change. Arguably, the latter provides greater overall benefits. First, the non-aspect-oriented model does not indicate where the crosscutting occurs, therefore requiring the locations to be found before they can be changed. This search-and-update approach is a very error-prone task. Second, the aspect-oriented model allows one to reason about the composition of a crosscutting concern as it is explicitly modeled. For example, changes to the number and location of pointcut matches may be brought to the attention of the modeler.

9.5.2 YKeyK System

The comparison of URN and AoURN models of the YKeyK system is based on three models described in Section 6.8 and 9.3.

Model-based Evaluation

Table 13 shows the results for the metrics applied to the three models YKeyK.A, YKeyK.B, and YKeyK.C with the best results highlighted with a grey background. In general, the lower the result, the better it is. Not all results, however, make sense, considering the pathological situation of YKeyK.A with only one goal graph and one map in the model as demonstrated by the extremely high result for the AvNME metric. CBMU results in 0 because coupling by definition requires at least two model units, which is not the case for YKeyK.A (VS is 1). A similar reasoning applies to LCOKME. Last but not least, CDMU counts model units and since there is only one model unit to count, the best result is achieved by default for each concern. Therefore, the artificially low results of the metrics CDMU, CBMU, LCOKME, and VS for YKeyK.A must not be taken into account.

The AoURN model of YKeyK.C performs significantly better than the URN models of YKeyK.A and YKeyK.B except in the following three cases. First, CDKME for YKeyK.A is better because the pointcut expressions in YKeyK.C are repeating elements from other goal graphs or maps. YKeyK.A, however, is not a viable approach as

there is only a single goal graph and map and the difference between the results of the CDKME metric is small. Despite the better results of CDKME for YKeyK.A, it cannot be concluded that YKeyK.A is a better approach than YKeyK.C.

Table 13 YKeyK Results of the Metrics

Metric	GRL			UCM			URN		
	A	B	C	A	B	C	A	B	C
CDMU	n/a (14)	42	21	n/a (3)	8	12	n/a (17)	50	33
CDKME	111	154	127	54	58	60	165	212	187
CDME	98	117	10	16	12	4	114	129	14
CBMU	n/a (0)	34	7	n/a (0)	4	7	n/a (0)	38	14
LCOKME	n/a (0)	88	20	n/a (0)	4	2	n/a (0)	92	22
VS	n/a (1)	13	35	n/a (1)	5	14	n/a (2)	18	49
NME	285	320	264	63	84	116	348	404	380
AvNME	285	25	12	63	17	11	174	27	13

Second, NME for YKeyK.A is better because YKeyK.C needs to add new model elements to express concerns and pointcut expressions. For the goal graphs in YKeyK.A, a large number of elements from crosscutting concerns have to be added explicitly to the GRL models. For the scenario models, however, no major crosscutting occurs because NFR concerns are not further refined with scenario models. Therefore, a similar effect could not be observed for the AoUCM model of YKeyK.C. Due to the non-viable nature of YKeyK.A, however, the results of NME again do not allow concluding that YKeyK.A is a better approach than YKeyK.C.

Third, VS for YKeyK.B is better because the AoURN-based approach introduces concerns and pointcut graphs to the model, explaining the difference in the results of the VS metric. The concerns, however, are used to group the AoURN model into more manageable sub-models which each contain a very small number of diagrams. Furthermore, the number of concerns in YKeyK.C (17) is very similar to the vocabulary size of YKeyK.B. This is further evidence for the trade-off between the complexity of individual model units and the overall number of modeling elements.

Looking at the individual UCM results, YKeyK.B is better than YKeyK.C for the CDMU, CDKME, and CBMU metrics. This result can be explained by the small amount of crosscutting that actually occurs in the UCM models for the YKeyK system. In a system with substantial crosscutting, these results are reversed, as can be observed for the goal models of the YKeyK system as well as in the CCCMS. CDMU is worse for the

UCM models in YKeyK.C, because concerns in YKeyK.C require several pointcut maps which are not justified by crosscutting in the UCM models of YKeyK.B. Similarly for CDKME, not enough model elements are required in the UCM models of YKeyK.B to model the crosscutting concern to justify the new modeling elements introduced by the pointcut expressions in YKeyK.C, again because there is very little crosscutting in the UCM models. Finally, CBMU is worse for the UCM models in YKeyK.C, because pointcut maps are coupled to the descriptions of aspectual properties. This is also not justified because of a lack of coupling due to crosscutting concerns in YKeyK.B models. Furthermore, the coupling between pointcut maps and the description of aspectual properties should not be considered as undesired.

In general, the results of the assessment of the YKeyK system confirm the results of the CCCMS, as both assessments exhibit the same characteristics and trends. The more crosscutting occurs in a model, the more it makes sense to structure URN models in an aspect-oriented way with AoURN.

Task-based Evaluation

In addition to the metrics, the most common update tasks are considered for all three approaches. These tasks cover adding, removing, or changing a non-functional requirement, use case, or stakeholder. For YKeyK.A, the tasks always involve finding, in the one large goal graph, all locations that require a change, and then updating them in a searching for needles-in-the-haystack approach. For YKeyK.B, the tasks require updating the goal graph for the relevant NFR or use case, and then finding the locations in the stakeholder goal graphs that also require an update and updating them. If a stakeholder is updated, all updates can be made on the stakeholder goal graph, but generally the updates are distributed over many different goal graphs. For YKeyK.C, the tasks require updates to the goal graphs, advice graphs, and pointcut graphs of a single concern (either the one encapsulating the non-functional requirement, the use case, or the stakeholder). In addition, changes to the number of matches of pointcut expressions can be monitored and reviewed, ensuring that the desired join points are still matched. Essentially, the results reflect the results obtained for the CCCMS models.

9.6. Summary

This chapter describes the assessment of AoURN models for the CCCMS and YKeyK systems against standard URN models of the systems based on metrics adapted from literature and a task-based evaluation. Overall, the results suggest that the AoURN models exhibit better modularity, reusability, and maintainability than the URN models. In general, the complexity of stakeholder goal graphs in GRL models is traded against the complexity of pointcut graphs in AoGRL. Similarly, the complexity of use case maps for use cases or NFRs is also traded against the complexity of pointcut maps. While pointcut expressions may be complex, they allow the complexity to be localized and are therefore more maintainable without hindering reusability. There is also a trade-off between the complexity of model units in URN models and the number of concepts and possibly modeling elements in AoURN models. While AoURN requires new concepts to be introduced into the model, the model units are generally less complex and smaller. Furthermore, the newly introduced concepts allow the model to be divided into more manageable sub-models and the relationships between the sub-models to be specified more explicitly. The larger the impact of a concern, i.e., the more crosscutting a concern is, the more pronounced are the positive results of the metrics assessment for AoURN models compared to URN models. The more a concern crosscuts the URN model, the more AoURN can reduce the complexity of model units without necessarily requiring increasingly complex pointcut expressions. With respect to scalability, the following five points arguably suggest that AoURN models are more scalable than URN models:

1. the reduced complexity of goal graphs and use case maps,
2. the ability to group goal graphs and use case maps into concerns,
3. the encapsulation provided by concerns,
4. the ability to use parameterized pointcut expressions/variables in AoURN, and
5. the simpler update tasks for AoURN.

It is possible to model all use cases, NFRs, and stakeholders of the case study systems in an aspect-oriented way with AoURN. As long as behavior can be specified for a concern, it can be modeled and encapsulated with AoUCM and applied to the rest of the model with aspect-oriented techniques. Even if behavior cannot be specified for a concern, it is still possible to capture the reasoning about the concern with AoGRL, encapsulate it in a

concern, and apply it to the overall goal model. At some later stage when more of the behavior is known for the concern, the AoGRL goal model of the concern may then be refined into an AoUCM scenario model.

In terms of threats to the validity of the performed assessment, a threat to internal validity is how concerns are derived from the system documentation. AoURN is not a technique that focuses on the detection of concerns. The decision to follow the general guidelines of considering use cases, NFRs, and stakeholders as concerns may have introduced bias into the assessment. However, the most pronounced results are observed for the various NFR concerns. Those highly-crosscutting concerns, however, are already identified by distinct sections in the CCCMS document, and only typical NFR concerns are required for the YKeyK system.

Furthermore, bias may have been introduced due to the experienced URN and AoURN modelers involved in the case study. The experience level combined with the facts that a) the URN and AoURN models have to be consistent, b) the content of the CCCMS document and hence the content of the models are not to be altered, and c) quite complete requirements for the YKeyK system are available from other case studies make it almost certain that high-quality URN and AoURN models are created for the case study systems. However, the resulting models may not reflect what less experienced modelers may have produced. Since AoURN arguably requires more training than URN, there is a greater possibility that less experienced modelers may produce AoURN models of lesser quality than URN models. This reduction in quality may cause the results to be less pronounced.

The risk that highly-experienced modelers may inadvertently introduce bias into the assessment is balanced to a certain extent by the exhaustive approach to the task-based evaluation since changes to each concern are investigated.

In terms of external validity, the CCCMS is a safety-critical application and it could be argued that the obtained results do not apply to other types of systems such as enterprise information systems. This threat is somewhat mitigated by the YKeyK system, an embedded system. Nevertheless, a problem description with less detailed and less well-specified information may lead to more divergent URN and AoURN models and possibly to different results.

In terms of the actual techniques for AoURN, the case study highlights the need for tailored, individual techniques for AoGRL and AoUCM. The fundamental differences in the structure of goal models and scenario models do not allow the same technique to be used effectively for both types of models. The highly interconnected nature of goal models makes it more amendable to a graph-transformation-like approach, where aspectual properties and pointcut expressions are specified on the same diagram. AoUCM, on the other hand, separates aspectual properties and pointcut expressions into two distinct diagrams for reusability reasons. In the CCCMS challenge problem, pointcut expressions are indeed reused by different concerns. This separation is possible for AoUCM, because scenario models are much more structured than goal models.

In some cases, the pointcut expressions required to encapsulate a concern with AoUCM can become quite complex. This is particularly true for concerns that require data-centric pointcut expressions such as the persistence NFR concern. Since the URN and AoURN frameworks do not contain sophisticated domain modeling capabilities, it is not possible to easily exploit type information of domain concepts or domain concept hierarchies when specifying pointcut expressions. Instead, concepts often have to be matched explicitly by enumerating all options. Enumerating all options quickly becomes cumbersome as experienced by modeling the persistence NFR concern with AoURN in the challenge problem. While it is possible to model data-centric concerns with AoURN, they are better served by techniques that have access to proper domain modeling facilities.

On the other hand, many concerns have very generic pointcut expressions that can be reused across application domains and are very resistant against changes to the base model. Many NFR concerns fall into this category (Figure 149 and Figure 150).

Finally, a crucial piece of the aspect-oriented modeling puzzle is the ability to deal with interactions between concerns. An aspect-oriented CCCMS model cannot be fully specified with techniques that cannot specify resolutions to concern dependencies and conflicts. AoURN's concern interaction graph deals with such interactions.

Chapter 10. Implementation

This chapter reports on the proof-of-concept implementation of features in support of aspect-oriented modeling with AoURN in the jUCMNav tool [62].

10.1. Prototype Overview

As the prototype's goal is not to provide a complete AoURN modeling environment, the most complex and critical features are the priority for the implementation. Therefore, the focus is on the matching and composition algorithms for AoUCM scenario models. While the matching algorithm is the same for AoUCM and AoGRL, the much simpler composition algorithm for AoGRL goal models is left for future work. The specification of AoUCM models, however, is fully supported as described in Chapter 6, as are simple concern management features. Some of the AoUCM specification features are provided courtesy of LavaBlast Software Inc. [75] and UniqueSoft LLC in the context of a major usability improvement project for jUCMNav. The specification of AoGRL models, on the other hand, requires only a simple update to the existing metadata feature in the jUCMNav tool and is therefore also left for future work.

The proof-of-concept implementation of the AoURN features is available from the AoURN home page at <http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/AoURN>. Figure 158 shows the concern management interface, allowing concerns to be updated and URN diagrams to be assigned to concerns. The Movie Points map is about to be assigned to the Movie Points concern. Figure 159 depicts the example for interleaving composition from Figure 90 on Page 136. The concern outline is seen on the left, the specification of the Movie Points concern in the middle, and the palette for AoURN model elements on the right. The Movie Points concern is about to be applied to the Order Movie concern. Figure 160 shows the result of the composition as shown in Figure 91 with all aspect markers applied to the Order Movie concern. The aspect markers can also be seen in the updated concern outline on the left. Also note how the plug-in

binding of the aspect marker after `sendMovie` is shown as a tooltip, because the cursor is placed over the aspect marker. The tooltip states that the aspect marker connects to the out-path of the pointcut stub `sendMovie` seen in Figure 159 and that the end point `point-sUsed` connects back to the aspect marker.



Figure 158 jUCMNav: Concern Management

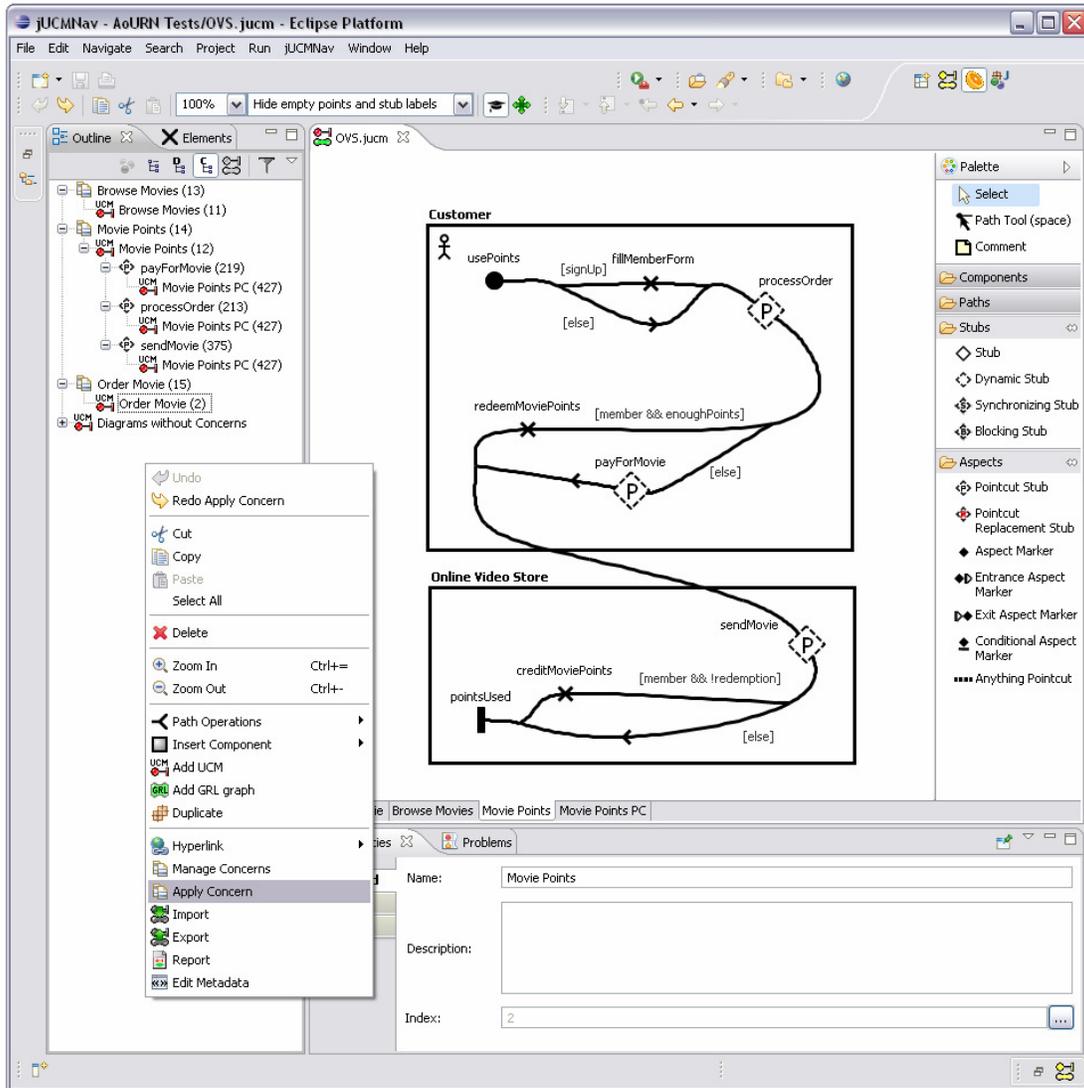


Figure 159 jUCMNav: Specification of Movie Points Concern

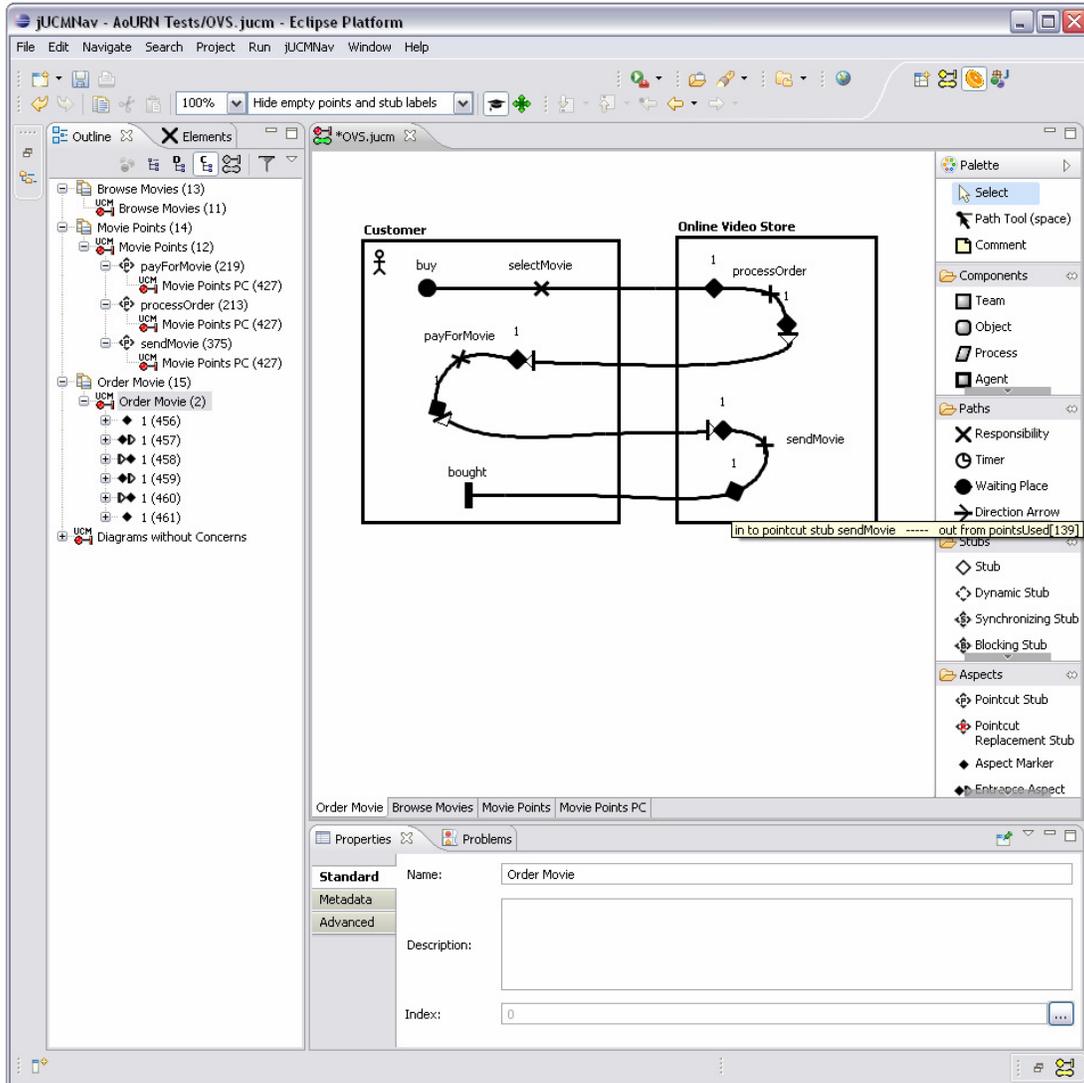


Figure 160 jUCMNav: Composition of Movie Points Concern

The implementation is tested with an ever-growing test suite comprised of more than 100 AoURN models representing close to 150 test cases. This test suite including its up-to-date fail/success status is also available from the AoURN home page. The proof-of-concept implementation provides evidence that all AoUCM examples in this thesis are feasible. The implementation supports the major features of AoUCM except for variables and the additional metadata added to support traversal of aspect-oriented UCM models. Several other smaller features are not supported as they build only on existing infrastructure available in the proof-of-concept implementation and do not require substantial new development. Table 14 gives an overview of those features. The implementation ensures

that aspect markers are added to the AoUCM model and that they are properly bound to aspect maps. The proof-of-concept implementation supports all composition rules including interleaving composition. However, while all other composition rules may be arbitrarily combined with each other, interleaved composition may not at this time. Simple combinations as shown in Figure 159 are supported but more complex combinations may not be. This is mostly a protection mechanism for the requirements engineer, because complex combinations with interleaved composition may lead to rather unexpected results.

In terms of computational complexity, the matching algorithm is of higher interest than the composition algorithm. The composition algorithm is quite straightforward, requiring a single pass for each match found by the matching algorithm. The matching algorithm, however, may have to deal with an explosion of potential matches. The number of potential matches depends on the content of the pointcut expression and the base model. If a pointcut expression is a simple sequence, then the AoURN base model can be matched linearly in one pass, i.e., for each element in the AoURN base model the pattern described by the pointcut expression is attempted to be matched exactly once. However, if a pointcut expression contains branch points with many branches, the anything pointcut element, or intentional elements with links to many other intentional elements, then the matching algorithm must compare permutations of the base model elements with the pointcut expression. Hence, this may require the examination of a significantly higher number of potential matches for each element in the AoURN base model, depending on the number of branches or GRL links of the base model element. In the worst case, all $n!$ permutations need to be considered. The computational complexity of the matching algorithm is therefore characterized by a combination of simple one-step tests for most AoURN model elements and $n!$ tests for some other AoURN model elements. However, usability issues related to performance, even with the non-optimized, proof-of-concept implementation, are not expected as matching and composition occurred instantaneously for all tests even for AoUCM models containing in excess of 30 maps. Typically, pointcut expressions as well as base models are sequences with maybe a few branch points and branches. Furthermore, AoUCM models with more than five branches for a branch point are rare. AoGRL models, on the other hand, will typically perform somewhat worse than

AoUCM models, because of their highly-interconnected nature, i.e., on average there are more intentional elements with many GRL links in an AoGRL model than there are branching points with many branches in an AoUCM model.

10.2. Feature List for Complete AoURN Modeling Environment

Table 14 lists the features for a complete AoURN modeling environment, indicating those that are provided fully (✓), partially (✓ with explanation), or not at all (✗) in the proof-of-concept implementation.

Table 14 Features for Complete AoURN Modeling Environment

AoURN Modeling Environment	Proof-of-Concept Implementation
Concern Management	
→ create, read, update, delete	✓
→ specify condition of concern	✗
→ specify abbreviation of concern	✗
→ assignment of model elements	✓ (assignment of diagrams only)
→ concern outline	✓ (outline shows only diagrams)
Concern Interaction Graph	
→ specification	✓ (except for metadata)
→ waves	✗ (only single aspect map with single pointcut expression applied at a time)
Specification of AoGRL Models	
→ aspect graph	✓
→ pointcut graph	✓ (except for metadata and links between same element)
→ composition rules	✓ (except for metadata)
Specification of AoUCM Models	
→ aspect map	✓
→ pointcut map	✓
→ composition rules	✓
Composition of AoGRL Models	
→ syntax-based matching	✗ (but this is the same algorithm as for AoUCM models)
→ semantics-based matching	✗
→ composition (aspect markers)	✗
→ AoViews	✗
Composition of AoUCM Models	
→ syntax-based matching	
→ UCM elements	✓ (all UCM elements are supported but not all matching criteria, no named start or end points at beginning or end of pointcut expression)
→ anything pointcut element	✓ (is supported except for matching against empty sequence and postprocessing)
→ variables	✗
→ semantics-based matching	
→ whitespace	✓
→ static stub	✓
→ dynamic stub	✗
→ synchronizing stub	✗

AoURN Modeling Environment	Proof-of-Concept Implementation
→ composition	
→ interleaving	✓ (limited combination with other composition rules)
→ other composition rules	✓
→ metadata	✗
→ variables	✗
→ regular aspect markers	✓
→ tunnel aspect markers	✓ (no postprocessing)
→ conditional aspect markers	✗ (regular aspect markers are used)
→ AoViews	
→ navigation	✓
→ highlighting	✗

10.3. Summary

This chapter describes the proof-of-concept implementation for the AoURN features in the jUCMNav tool and provides references for obtaining the implementation as well as its test suite. It also briefly discusses the computational complexity of the implementation. The computational complexity of the matching algorithm depends on the content of the pointcut expression and the base model. The higher the number of branches in a pointcut expression and the base model, the higher the number of anything pointcut elements, and the higher the number of intentional elements with many GRL links, the more permutations have to be matched by the matching algorithm. In the best case, the matching algorithm performs with linear complexity and in the worst case with $O(n!)$. However, in practice, this potential complexity is not an issue because the size of pointcut expressions is small and branching elements with a large number of branches are rare in typical AoUCM models. AoGRL models are expected to perform somewhat worse than AoUCM models because of their highly-interconnected nature. The composition algorithm, on the other hand, is of linear complexity in terms of the number of matches found by the matching algorithm.

The chapter concludes with a set of features for a complete AoURN modeling environment, which can be used as a roadmap for future improved support for AoURN in the jUCMNav tool.

Chapter 11. Conclusions

This chapter summarizes the thesis' contributions in Section 11.1 and discusses future work in Section 11.2.

11.1. Contributions

This thesis introduces the Aspect-oriented User Requirements Notation (AoURN), unifying goal-oriented, scenario-based, and aspect-oriented concepts in one framework for requirements engineering. The thesis explains how to specify aspect-oriented goal and scenario models with AoURN, how the pattern matching process functions for AoURN specifications, and how the composition of AoURN models unfolds. In support of these features, the abstract syntax, the concrete syntax, and the semantics of URN are extended with aspect-oriented concepts. AoURN is the only aspect-oriented technique for goal and scenario models for requirements engineering activities that builds on an existing international standard for requirements engineering and utilizes a flexible, exhaustive, semantics-based aspect matching and composition technique appropriate for the model type to which it is applied. As AoURN composition rules are expressed by URN itself, they are not limited by a particular composition language. The thesis also clarifies the relationship of goal and scenario aspects within AoURN. Based on a quantitative assessment of AoURN and URN models, AoURN models exhibit greater modularity, maintainability, and reusability, and suggest better scalability than equivalent URN models. The more crosscutting a concern is, the more pronounced the benefits of AoURN are with respect to the aforementioned qualities.

AoURN's advanced matching and composition techniques require the clarification of the semantics of URN. Most of the proposed non-aspect-oriented changes to URN are already incorporated into the last version of the URN standard. Of the aspect-oriented concepts discussed in this thesis, the concern concept is also part of the latest version of the URN standard to enable the grouping of URN model elements.

The matching and composition mechanisms of AoURN are enhanced based on the semantics of URN, exploiting semantic equivalences of the language. This semantics-based mechanism is an extensible approach as new equivalences may be added to the matching and composition algorithms. AoURN focuses on equivalences related to hierarchical structuring.

AoURN's composition rules go well beyond typical before/after/around operators and include concurrent, looped, as well as interleaved composition. Even potentially complex combinations of interleaved and other composition rules are supported, but may require additional processing of the AoURN model specifications for composition, as does the composition of results of the semantics-based matching algorithm. On the other hand, the required extensions for the matching algorithm to support interleaving and semantics-based matching are rather straightforwardly integrated into the basic matching algorithm.

The evaluation of AoURN rests on three pillars which are discussed in this thesis: a qualitative comparison of aspect-oriented modeling techniques for goal and scenario models based on qualitative factors, a quantitative assessment of AoURN and URN models based on metrics for separation of concerns, coupling, cohesion, and size as well as various update tasks, and a proof-of-concept implementation of AoURN's matching and composition algorithms.

11.2. Future work

Future work can be broadly grouped into five categories:

- expanding AoURN concepts to the analysis capabilities of URN and investigating the applicability of advanced URN research to AoURN,
- exploring the connections of aspect-oriented models at different levels of abstraction, starting from AoURN models at the requirements level,
- investigating the usability of AoURN through empirical studies,
- strengthening the formal foundation of URN and hence of AoURN, and
- incorporating further AoURN concepts into the official URN standard.

Applying the existing URN analysis techniques for trade-off analysis as well as scenario verification and validation efficiently to AoURN models requires the analysis information to also be organized in an aspect-oriented way [97]. Furthermore, the applicability of advanced URN research to AoURN should be investigated (e.g., in the areas of feature interaction [97], business process modeling [121], performance analysis, and software product lines [94]). Note that some of these areas, and especially business process improvement, can highly benefit from aspects that combine goals and scenarios at the same time [87][120].

The benefits of aspect-oriented modeling may be multiplied with the establishment of a fully aspect-oriented, end-to-end process, starting at the requirements level, e.g., with AoURN, and proceeding all the way down the abstraction ladder to aspect-oriented implementations of systems. Therefore, the connections and traceability of aspect-oriented models at different levels of abstraction, starting from AoURN models at the requirements level, should be explored. The exploration of relationships between AoURN models and aspect-oriented designs has already begun [84]. Major research questions in this area are how can AoURN's advanced composition rules be supported by further downstream modeling activities and how can AoGRL's reasoning framework be applied to guide the selection of concerns at lower levels of abstraction.

Quantitative assessments show that AoURN has advantages over URN in terms of modularity, maintainability, reusability, and scalability. However, further empirical studies in the form of usability studies should be undertaken to better understand the complexity of AoURN models as perceived by requirements engineers and to experiment with various different ways of presenting to requirements engineers the concepts that are required for AoURN as discussed in this thesis. Industrial studies could also shed more light on the scalability of the graphical approach of AoURN. The jUCMNav tool should first be updated to a more complete modeling environment for AoURN to support such studies.

In addition, in the context of such usability assessments of AoURN, some of the current restrictions for AoURN models may also be addressed and further extensions investigated. For example, it is currently not allowed for interleaved composition to change the partial ordering of the scenarios or to use different pointcut maps for its pointcut

stubs, as these variations of interleaving may lead to undesired composition results. It is also not allowed to reuse base model elements on a map other than the top level map of a concern as this requires changes to URN's definition of component plug-in bindings, or to define variables for the anything pointcut element as this potentially involves complex layout issues for the reused path segment and may lead to invalid UCM models. At this point, pointcut expressions may not consist of only components or actors, and the deletion of base elements is indicated in AoUCM models but not in AoGRL models. Furthermore, plug-in maps in pointcut expressions are currently not matched as plug-in maps, because allowing such matches may lead to similar issues with aspect markers at different map levels as can be observed for the results of semantics-based matching. Finally, dynamic information such as the values of GRL evaluations or the content of UCM scenario variables could be taken into account by the AoURN matching algorithm, likely leading to a more complete categorization and catalogue of crosscutting patterns that should be supported at the requirements level (e.g., before, after, around, alternative, concurrent, looped, interleaved, etc.).

Furthermore, again in the context of usability assessments of AoURN, one may also want to experiment with generic graph transformation and graph pattern matching systems as well as other generic weaving techniques. While such generic tools may be used to implement AoURN's matching and composition engine, enabling good visualization of the result to the requirements engineer would remain a challenge for UCM models and to a certain extent also for GRL models.

An issue more with URN itself and not directly with AoURN is the formal foundation of the language, which should be strengthened. URN and AoURN could also benefit from tighter integration with approaches for domain modeling or conceptual modeling. Goal modeling with URN and AoURN could also be improved by greater support for hierarchical modeling of the relationships between actors and intentional elements at different levels in the goal tree.

Finally, incorporating more AoURN concepts into the official URN standard should also be pursued. The possible inclusion of AoURN into the next version of the URN standard was discussed for the first time by ITU-T in September 2009. It was however suggested that it is premature to decide on its inclusion since URN itself is still a

rather new standard and more experience with AoURN is first required. This thesis represents an important step in demonstrating AoURN's potential in that context.

Glossary

The glossary lists all new concepts introduced by the Aspect-oriented User Requirements Notation (AoURN) and Use Case Maps (UCM) 2.0, giving definitions for each concept and referencing the section where the concept is described in more detail as well as all related concepts in the glossary. For each concept, it is also stated whether it belongs to the Aspect-oriented and Goal-oriented Requirement Language (AoGRL), Aspect-oriented Use Case Maps (AoUCM), AoURN (i.e., AoGRL and AoUCM), or UCM 2.0.

abort scope – a set of paths defined by the location of the abort start point that are aborted when the abort start point is traversed (UCM 2.0, Section 3.7.1);
see also [abort start point](#), [explicit cancellations and exceptions](#), [failure condition](#), [failure expression](#), [failure point](#), [failure start point](#), [failure variable](#), [guarding condition](#), [implicit cancellations and exceptions](#).

abort start point – a failure start point that additionally aborts any active scenario in the abort scope (UCM 2.0, Section 3.7.1);
see also [abort scope](#), [explicit cancellations and exceptions](#), [failure condition](#), [failure expression](#), [failure point](#), [failure start point](#), [failure variable](#), [guarding condition](#), [implicit cancellations and exceptions](#).

anything pointcut element – a path node that is used on a pointcut map to match any sequence of UCM base elements including an empty sequence (AoUCM, Section 6.5.2);
see also [pointcut map](#), [pointcut variable](#).

anytype pointcut element – an annotation of a single intentional element or single GRL link on a pointcut graph that matches a GRL intentional element or a GRL link regardless of the type of intentional element or link, respectively (AoGRL, Section 6.5.2);
see also [pointcut deletion marker](#), [pointcut graph](#), [pointcut marker](#).

AoView – a visualization of the aspectual properties of an aspect inserted at the insertion point identified by an aspect marker (AoURN, Sections 6.1.4 and 6.4);
see also [aspect marker](#), [aspectual properties](#), [insertion point](#).

aspect – a crosscutting concern (AoURN, Section 6.1.5);
see also [concern](#).

aspect diagram – either an [aspect graph](#) or an [aspect map](#) (AoURN, Section 6.1.2).

- aspect graph* – a GRL graph that defines aspectual properties in an AoGRL goal model of an aspect (AoGRL, Sections 6.1.2 and 6.2.2);
see also [aspectual properties](#).
- aspect map* – a map that defines aspectual properties and composition rules in an AoUCM scenario model of an aspect (AoUCM, Sections 6.1.2 and 6.3.2);
see also [aspectual properties](#), [composition rule](#), [pointcut stub](#), [pointcut variable](#).
- aspect marker* – identifies an insertion point for aspectual properties of an aspect in the base model (AoURN, Sections 6.1.4, 6.2.4, and 6.3.4);
see also [AoView](#), [aspect marker group](#), [aspect stub](#), [base model](#), [conditional aspect marker](#), [insertion point](#), [tunnel entrance aspect marker](#), [tunnel exit aspect marker](#).
- aspect marker group* – a set of aspect markers in the AoUCM model that allow an aspect map to be entered through one of the aspect markers in the set and exited through another aspect marker in the set (AoUCM, Section 6.5.4);
see also [aspect marker](#), [conditional aspect marker](#), [tunnel entrance aspect marker](#), [tunnel exit aspect marker](#).
- aspect stub* – an aspect marker in the AoUCM model is represented as an aspect stub, a special kind of static or dynamic stub that links to the portion of the aspect map that needs to be added at the insertion point (AoUCM, Section 6.3.4);
see also [aspect map](#), [aspect marker](#), [insertion point](#).
- aspectual properties* – the aspect-specific intentions, behaviors, and structures specified on aspect diagrams (AoURN, Section 6.1.2);
see also [AoView](#), [aspect graph](#), [aspect map](#), [pointcut graph](#).
- base* or *base model* – the part of an AoURN model that is matched against the pointcut expression of an aspect, i.e., the complete AoURN model except for the diagrams of the aspect that is to be matched, the diagrams of the concerns that are to be applied in the next waves as defined in the concern interaction graph, all pointcut expressions, and the concern interaction graph (AoURN, Section 7.1);
see also [aspect marker](#), [concern interaction graph](#), [mapping](#), [match](#), [pointcut expression](#), [wave](#).
- blocking* – a synchronizing stub that cannot be entered another time along the same in-path before all initiated plug-in maps have completed (UCM 2.0, Section 3.4);
see also [cancelling](#), [replication factor](#), [synchronizing stub](#), [synchronization threshold](#).
- cancelling* – a synchronizing stub that aborts all remaining active plug-in maps of the same visit when the required number of plug-in maps have arrived at each out-path of the stub for the visit (UCM 2.0, Section 3.4);
see also [blocking](#), [replication factor](#), [synchronizing stub](#), [synchronization threshold](#), [visit](#).
- component plug-in binding* – defines for a particular stub and plug-in map the component on the map of the stub that replaces a component identified by the keyword parent on the plug-in map (UCM 2.0, Section 3.6.1);
see also [parent](#), [responsibility plug-in binding](#).

- composition rule* – defines how the aspectual properties of an aspect are to be applied to the join points matched by the pointcut expressions of the aspect (AoURN, Sections 6.1.4, 6.2.4, and 6.3.4);
see also [aspectual properties](#), [aspect map](#), [interleaved composition rule](#), [join point](#), [pointcut expression](#), [pointcut graph](#), [pointcut stub](#).
- concern* – an organizational construct that contains all URN diagrams required to describe a unit of interest to a requirements engineer, e.g., a use case, feature, non-functional requirement, stakeholder, etc. (AoURN, Section 6.1.5);
see also [aspect](#).
- concern interaction graph* – a specialized GRL graph that documents conflicts and dependencies between concerns (AoURN, Section 6.1.5);
see also [concern](#), [precedence rule](#), [wave](#).
- condition* – see [failure condition](#) for condition of a failure point, see [guarding condition](#) for conditions of failure start points and abort start point, see [mapping](#) for condition of a mapping.
- conditional aspect marker* – an aspect marker in the AoUCM model that additionally indicates that the scenario may or may not continue with the base elements following the conditional aspect marker (AoUCM, Section 6.5.4);
see also [aspect marker](#), [tunnel entrance aspect marker](#), [tunnel exit aspect marker](#).
- explicit cancellations and exceptions* – an approach for modeling cancellations and exceptions that explicitly specifies failure points in the UCM model (UCM 2.0, Section 3.7.1);
see also [abort scope](#), [abort start point](#), [failure condition](#), [failure expression](#), [failure point](#), [failure start point](#), [failure variable](#), [guarding condition](#), [implicit cancellations and exceptions](#).
- failure condition* – a Boolean expression associated with a failure point that describes under what conditions a failure occurs in a scenario (UCM 2.0, Section 3.7.1);
see also [abort scope](#), [abort start point](#), [explicit cancellations and exceptions](#), [failure expression](#), [failure point](#), [failure start point](#), [failure variable](#), [guarding condition](#), [implicit cancellations and exceptions](#).
- failure expression* – a variable assignment associated with a failure point that indicates which failure has occurred by setting a failure variable to true (UCM 2.0, Section 3.7.1);
see also [abort scope](#), [abort start point](#), [explicit cancellations and exceptions](#), [failure condition](#), [failure point](#), [failure start point](#), [failure variable](#), [guarding condition](#), [implicit cancellations and exceptions](#).
- failure point* – a path node that indicates explicitly the location of a failure in a scenario by specifying a failure condition and a failure expression that sets a failure variable (UCM 2.0, Section 3.7.1);
see also [abort scope](#), [abort start point](#), [explicit cancellations and exceptions](#), [failure condition](#), [failure expression](#), [failure start point](#), [failure variable](#), [guarding condition](#), [implicit cancellations and exceptions](#).

- failure start point* – a path node that specifies a guarding condition and indicates the start of a path executed in response to a failure (UCM 2.0, Section 3.7.1);
see also [abort scope](#), [abort start point](#), [explicit cancellations and exceptions](#), [failure condition](#), [failure expression](#), [failure point](#), [failure variable](#), [guarding condition](#), [implicit cancellations and exceptions](#).
- failure variable* – a scenario variable that describes a failure and is set by failure expressions (UCM 2.0, Section 3.7.1);
see also [abort scope](#), [abort start point](#), [explicit cancellations and exceptions](#), [failure condition](#), [failure expression](#), [failure point](#), [failure start point](#), [guarding condition](#), [implicit cancellations and exceptions](#).
- guarding condition* – a Boolean expression associated with a failure start point or abort start point that indicates to which failure to respond with the help of failure variables (UCM 2.0, Section 3.7.1);
see also [abort scope](#), [abort start point](#), [explicit cancellations and exceptions](#), [failure condition](#), [failure expression](#), [failure point](#), [failure start point](#), [failure variable](#), [implicit cancellations and exceptions](#).
- implicit cancellations and exceptions* – an approach for modeling cancellations and exceptions that defines the locations of failures with the help of scenario definitions (UCM 2.0, Section 3.7.2);
see also [abort scope](#), [abort start point](#), [explicit cancellations and exceptions](#), [failure condition](#), [failure expression](#), [failure point](#), [failure start point](#), [failure variable](#), [guarding condition](#).
- insertion point* – associated with exactly one join point, identifies precisely where aspectual properties of an aspect may be inserted (AoURN, Sections 6.2.1 and 6.3.1);
see also [aspect marker](#), [aspectual properties](#), [insertion point arrow](#), [join point](#).
- insertion point arrow* – identifies, during the composition process, the insertion point to be used when applying an aspect (AoUCM, Section 7.2.3);
see also [insertion point](#).
- interleaved composition rule* – a composition rule that merges two independent scenarios while respecting the partial ordering of each scenario (AoUCM, Section 6.5.6);
see also [composition rule](#).
- join point* – a location in the URN model with one or more insertion points which may be modified by an aspect (AoURN, Sections 6.1.1, 6.2.1, and 6.3.1);
see also [composition rule](#), [insertion point](#), [join point model](#), [mapping](#), [pointcut expression](#).
- join point model* – specifies all possible locations in the AoURN model, i.e., the join points, which may be modified by an aspect (AoURN, Sections 6.1.1, 6.2.1, and 6.3.1);
see also [aspect](#), [join point](#).
- key model element* – intentional element, responsibility, or stub (AoURN, Section 9.4);
see also [model element](#), [model unit](#).

- local end point* – an end point that cannot be used in plug-in bindings of stubs including aspect stubs (UCM 2.0, AoUCM, Sections 3.3.5 and 6.5.5);
see also [aspect stub](#).
- local start point* – a start point that cannot be used in plug-in bindings of stubs including aspect stubs (UCM 2.0, AoUCM, Sections 3.3.5 and 6.5.5);
see also [aspect stub](#).
- mapping* – a result of the matching algorithm that establishes a link between an element in the pointcut expression and a join point in the base model, a mapping may be conditional depending on the values of variables, component plug-in bindings, and responsibility plug-in bindings in the base model (AoURN, Sections 6.1.3 and 7.1);
see also [base model](#), [join point](#), [match](#), [pointcut expression](#).
- match* – a set of mappings that, if successful, matches a complete pointcut expression against the base model (AoURN, Section 7.1);
see also [base model](#), [mapping](#), [pointcut expression](#).
- model element* – any AoURN modeling element (AoURN, Section 9.4);
see also [key model element](#), [model unit](#).
- model unit* – goal graph or use case map (AoURN, Section 9.4);
see also [key model element](#), [model element](#).
- parent* – a prefix for the name of a component or responsibility on a plug-in map that indicates that a component or responsibility plug-in binding, respectively, needs to be specified for this component or responsibility, respectively (UCM 2.0, Sections 3.6.1 and 3.6.2);
see also [component plug-in binding](#), [responsibility plug-in binding](#).
- pointcut deletion marker* – a pointcut marker that additionally indicates that the matched base element is to be removed from the AoGRL model when the aspect is applied (AoGRL, Section 6.2.3);
see also [anytype pointcut element](#), [pointcut marker](#), [pointcut graph](#).
- pointcut diagram* – either a [pointcut graph](#) or a [pointcut map](#) (AoURN, Section 6.1.3).
- pointcut expression* – a pattern defined for an aspect that is matched against the base model to find the join points modified by the aspect (AoURN, Section 6.1.3);
see also [base model](#), [composition rule](#), [join point](#), [mapping](#), [match](#), [pointcut graph](#), [pointcut map](#).
- pointcut graph* – a GRL graph that specifies a pointcut expression and composition rules as well as aspectual properties for the goal model of an aspect (AoGRL, Sections 6.1.3 and 6.2.3);
see also [anytype pointcut element](#), [aspectual properties](#), [composition rule](#), [pointcut deletion marker](#), [pointcut expression](#), [pointcut marker](#).
- pointcut map* – a map that specifies a pointcut expression for the scenario model of an aspect (AoUCM, Sections 6.1.3 and 6.3.3);
see also [anything pointcut element](#), [pointcut expression](#), [pointcut variable](#).

pointcut marker – an annotation of an intentional element, actor, or GRL link on a pointcut graph that indicates that the model element is part of a pointcut expression of an aspect and hence matched against the base model (AoGRL, Section 6.2.3); see also [anytype pointcut element](#), [base model](#), [pointcut deletion marker](#), [pointcut expression](#), [pointcut graph](#).

pointcut stub – a kind of stub that is used on aspect maps as a placeholder for the matched base elements and to define composition rules (AoUCM, Section 6.3.3); see also [aspect map](#), [composition rule](#), [replacement pointcut stub](#).

pointcut variable – a component or responsibility identified by the prefix \$ on a pointcut map or aspect map that allows the matched base element to be reused by the aspect on an aspect map (AoUCM, Section 6.5.3); see also [anything pointcut element](#), [aspect map](#), [pointcut map](#).

precedence rule – determines for a pair of concerns the order in which they are applied to the AoURN model (AoURN, Section 6.1.5); see also [concern](#), [concern interaction graph](#), [wave](#).

replacement pointcut stub – a pointcut stub that additionally indicates that the matched base elements are to be removed from the AoUCM model when the aspect is applied (AoUCM, Section 6.3.4); see also [pointcut stub](#).

replication factor – an integer number that specifies how many instances of a plug-in map of a dynamic or synchronizing stub are to be executed in parallel (UCM 2.0, Section 3.4); see also [blocking](#), [cancelling](#), [synchronizing stub](#), [synchronization threshold](#).

responsibility plug-in binding – defines for a particular stub and plug-in map the responsibility definition in the UCM model that replaces a responsibility identified by the keyword parent on the plug-in map (UCM 2.0, Section 3.6.2); see also [component plug-in binding](#), [parent](#).

singleton map – a map that cannot be instantiated multiple times and hence exists only once in the UCM model (UCM 2.0, Section 3.5.1).

synchronization threshold – an integer expression greater than zero that may be defined for an out-path of a synchronizing stub to override the default behavior by specifying the number of required arrivals at the out-path before the scenario is allowed to continue past the stub (UCM 2.0, Section 3.4); see also [blocking](#), [cancelling](#), [replication factor](#), [synchronizing stub](#).

synchronizing stub – a dynamic stub whose plug-in maps are executed in parallel and have to synchronize before the scenario is allowed to continue past the stub (UCM 2.0, Section 3.4); see also [blocking](#), [cancelling](#), [replication factor](#), [synchronization threshold](#).

tunnel aspect marker – either a [tunnel entrance aspect marker](#) or a [tunnel exit aspect marker](#) (AoURN, Section 6.5.4).

tunnel entrance aspect marker – an aspect marker paired with a tunnel exit aspect marker in the AoUCM model that additionally indicates that the scenario does not con-

- tinue with the base elements following the tunnel entrance aspect marker but at the location identified by the tunnel exit aspect marker, thus replacing base elements with aspectual properties (AoUCM, Section 6.5.4);
see also [aspect marker](#), [aspectual properties](#), [conditional aspect marker](#), [tunnel exit aspect marker](#).
- tunnel exit aspect marker* – an aspect marker paired with a tunnel entrance aspect marker in the AoUCM model that additionally indicates the end of a replacement of base elements with aspectual properties (AoUCM, Section 6.5.4);
see also [aspect marker](#), [aspectual properties](#), [conditional aspect marker](#), [tunnel entrance aspect marker](#).
- variable* – see [pointcut variable](#).
- visit* – the number of times an in-path of a stub has been visited (UCM 2.0, Section 3.5.2);
see also [cancelling](#).
- wave* – a group of concerns that is applied to the AoURN model during the same phase in the matching and composition process, ensuring that each concern in the group is matched against an AoURN model to which the same concerns are applied (AoURN, Section 7.2);
see also [concern](#), [concern interaction graph](#), [precedence rule](#).
- whitespace* – semantically insignificant model elements that exist for visualization purposes only (e.g., direction arrow, empty points, and connected end and start points in UCM models) (AoUCM, Section 7.1.2).

References

- [1] Abdelaziz, T., Elammari, M., and Unland, R.: “Visualizing a Multiagent-Based Medical Diagnosis System Using a Methodology Based on Use Case Maps”. Lindemann-v. Trzebiatowski, G., Denzinger, J., Timm, I.J., and Unland, R. (Editors), *Multiagent System Technologies*, Springer, LNCS 3187, pp. 198–212 (September 2004)
- [2] Alencar, F., Castro, J., Moreira, A., Araújo, J., Silva, C., Monteiro, C., Ramos, R., and Mylopoulos, J.: “Simplifying i* Models”. *17th International Workshop on Agent-Oriented Information Systems (AOIS-2007)*, Trondheim, Norway (June 2007)
- [3] Alencar, F., Castro, J., Moreira, A., Araújo, J., Silva, C., Ramos, R., and Mylopoulos, J.: "Integration of Aspects with i* Models". Kolp, M., Henderson-Sellers, B., Mouratidis, H., Garcia, A., Ghose, A., and Bresciani, P. (Editors), *Agent-Oriented Information Systems IV*, Springer, LNCS 4898, pp. 183–201 (2008)
- [4] Alencar, F., Moreira, A., Araújo, J., Castro, J., Ramos, R., and Silva, C.: “Proposal to deal with the Complexity of i* Models with Aspects”. *1st International Conference on Research Challenges in Information Science (RCIS'07)*, Quarzazate, Marocco (April 2007)
- [5] Alencar, F., Moreira, A., Araújo, J., Castro, J., Silva, C., and Mylopoulos, J.: “Towards an Approach to Integrate i* with Aspects”. *8th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2006)* at CAiSE'06, Luxembourg (June 2006)
- [6] Alencar, F., Moreira, A., Araújo, J., Castro, J., Silva, C., and Mylopoulos J.: “Using Aspects to Simplify i* Models”. *14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA (September 2006)
- [7] Alsumait, A.: *User Interface Requirements Engineering: A Scenario-Based Framework*. Ph.D. thesis, Concordia University, Canada (August 2004)
- [8] Amyot, D. and Eberlein, A.: “An Evaluation of Scenario Notations and Construction Approaches for Telecommunication”. *Telecommunications Systems Journal*, Vol. 24(1), pp. 61–94 (September 2003)
- [9] Amyot, D. and Logrippo, L.: “Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System”. *Computer Communication*, Vol. 23(12), pp. 1135–1157 (July 2000)

- [10] Amyot, D. and Mussbacher, G.: “Development of Telecommunications Standards and Services with the User Requirements Notation”. *Workshop on ITU System Design Languages 2008*, Geneva, Switzerland (September 15-16, 2008)
- [11] Amyot, D. and Yan, J.B.: “Flexible Verification of User-Defined Semantic Constraints in Modelling Tools”. *18th International Conference of Computer Science and Software Engineering (CASCON 2008)*, Toronto, Canada (October 2008)
- [12] Amyot, D., Becha, H., Bræk, R., and Rossebø, J.E.Y.: “Next Generation Service Engineering”. *ITU-T Innovations in NGN Kaleidoscope Conference*, Geneva, Switzerland (May 2008)
- [13] Amyot, D., Charfi, L., Gorse, N., Gray, T., Logrippo, L., Sincennes, J., Stepien, B., and Ware, T.: “Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS”. *Feature Interactions in Telecommunications and Software Systems VI*, Glasgow, Scotland, UK, IOS Press, pp. 274–289 (May 2000)
- [14] Amyot, D., Horkoff, J., Gross, D., and Mussbacher, G.: “A Lightweight GRL Profile for i* Modeling”. *3rd International Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGiM 2009)*, Gramado, Brazil (November 2009). Heuser, C.A. and Pernul, G. (Editors), *Advances in Conceptual Modeling - Challenging Perspectives*, Springer, LNCS 5833, pp. 254–264 (2009)
- [15] Amyot, D., Mussbacher, G., and Mansurov, N.: “Understanding Existing Software with Use Case Map Scenarios”. *3rd SDL and MSC Workshop (SAM02)*, Aberystwyth, Wales, UK, Springer, LNCS 2599, pp. 124–140 (June 2002)
- [16] Amyot, D., Roy, J.-F., and Weiss, M.: “UCM-Driven Testing of Web Applications”. Prinz, A., Reed, R., and Reed, J. (Editors), *SDL 2005: Model Driven*, Springer, LNCS 3530, pp. 247–264 (June 2005)
- [17] Amyot, D., Weiss, M., and Logrippo L.: “UCM-Based Generation of Test Purposes”. *Computer Networks*, Elsevier, Vol. 49(5), pp. 643–660 (December 2005)
- [18] Amyot, D.: “Introduction to the User Requirements Notation: Learning by Example”. *Computer Networks*, Elsevier, Vol. 42(3), pp. 285–301 (21 June 2003)
- [19] Andrade, R.: “Applying Use Case Maps and Formal Methods to the Development of Wireless Mobile ATM Networks”. *Lfm2000: 5th NASA Langley Formal Methods Workshop*, Williamsburg, Virginia, USA, pp. 151–162 (June 2000)
- [20] *AOSD Community Wiki – Research Projects & Tools for Developers*. http://aosd.net/wiki/index.php?title=Research_Projects (accessed November 2010), http://aosd.net/wiki/index.php?title=Tools_for_Developers (accessed November 2010)
- [21] Araújo J. and Coutinho, P.: “Identifying Aspectual Use Cases Using a Viewpoint-Oriented Requirements Method”. *Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design*, Workshop of the 2nd International Conference on Aspect-Oriented Software Development (AOSD), Boston, Massachusetts, USA (March 2003)

- [22] Araújo, J. and Moreira, A.: “An Aspectual Use Case Driven Approach”. *VIII Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2003)*, Alicante, Spain (November 2003)
- [23] Araújo, J., Whittle, J., and Kim, D.: “Modeling and Composing Scenario-Based Requirements with Aspects”. *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04)*, Kyoto, Japan, IEEE Computer Society Press, pp. 58–67 (Sep. 2004)
- [24] *AspectJ web site*. <http://www.eclipse.org/aspectj/> (accessed November 2010)
- [25] Barros, J.-P. and Gomes, L.: “Toward the Support for Crosscutting Concerns in Activity Diagrams: a Graphical Approach”. *Workshop on Aspect-Oriented Modeling* (held with UML 2003), San Francisco, California, USA (October 2003)
- [26] Basili, V., Caldiera, G., and Rombach, H.: “The Goal Question Metric Approach”. *Encyclopedia of Software Engineering*, John Wiley & Sons, Inc., Vol. 2, pp. 528–532 (September 1994)
- [27] Becha, H., Mussbacher, G., and Amyot, D.: “Modeling and Analyzing Non-Functional Requirements in Service Oriented Architecture with the User Requirements Notation”. Milanovic, N. (Editor), *Non-functional Properties in Service Oriented Architecture: Requirements, Models and Methods*, IGI Global (to appear)
- [28] Billard, E.A.: “Operating system scenarios as Use Case Maps”. *4th International Workshop on Software and Performance (WOSP 2004)*, Redwood Shores, California, USA, pp. 266–277 (January 2004)
- [29] Braem, M., Gybels, K., Kellens, A., and Vanderperren, W.: “Inducing Evolution-Robust Pointcuts”. *Second International ERCIM Workshop on Software Evolution (EVOL 2006)*, Lille, France (April 2006)
- [30] Brito, I., Vieira, F., Moreira, A., and Ribeiro, R.: "Handling Conflicts in Aspectual Requirements Compositions". Rashid, A. and Aksit, M. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) III*, Springer, LNCS 4620, pp. 144–166 (2007)
- [31] Buhr, R.J.A. and Casselman, R.S.: *Use Case Maps for Object-Oriented Systems*. Prentice-Hall (1996)
- [32] Buhr, R.J.A.: “A Possible Design Notation for Aspect Oriented Programming”. *Aspect-Oriented Programming Workshop at ECOOP'98*, Brussels, Belgium (July 1998)
- [33] Buhr, R.J.A.: “Use Case Maps as Architectural Entities for Complex Systems”. *IEEE Transactions on Software Engineering*, Vol. 24(12), pp. 1131–1155 (December 1998)
- [34] *Business Process Execution Language for Web Services (BPEL4WS) 1.1*. <http://www.ibm.com/developerworks/library/specification/ws-bpel> (accessed November 2010)

- [35] CCCMS: *Complete URN Model and AoURN Model Files*. <http://www.site.uottawa.ca/~damyot/pub/CCCMS-TAOSD> (accessed November 2010)
- [36] Chitchyan, R. et al.: *Survey of Analysis and Design Approaches*. AOSD-Europe Report ULANC-9 (May 2005); <http://www.aosd-europe.net/deliverables/d11.pdf> (accessed November 2010)
- [37] Chitchyan, R., Rashid, A., Rayson, P., and Waters, R.W.: "Semantics-based Composition for Aspect-Oriented Requirements Engineering". *6th International Conference on Aspect-Oriented Software Development (AOSD'07)*, Vancouver, Canada, ACM, pp. 36–48 (March 2007)
- [38] Chung, L., Nixon, B.A., Yu, E., and Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, Dordrecht, USA, (2000)
- [39] Clarke, S. and Baniassad, E.: *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley (2005)
- [40] Cottenier, T., van den Berg, A., and Elrad, T.: "Joinpoint Inference from Behavioral Specification to Implementation". Ernst, E. (Editor), *ECOOP 2007*, Springer, LNCS 4609, pp. 476–500 (2007)
- [41] Cui, Z., Wang, L., Li, X., and Xu, D.: "Modeling and Integrating Aspects with UML Activity Diagrams". *2009 ACM Symposium on Applied Computing (SAC'09)*, Honolulu, Hawaii, USA, ACM, pp. 430–437 (2009)
- [42] da Silva, L. F.: "An Aspect-Oriented Approach to Model Requirements". *Doctoral Consortium at the 13th International Requirements Engineering Conference*, Paris, France (August-September 2005)
- [43] de Bruin, H. and van Vliet, H.: "Quality-Driven Software Architecture Composition". *Journal of Systems and Software*, Vol. 66(3), pp. 269–284 (15 June 2003)
- [44] Deubler, M., Meisinger, M., Rittmann, S., and Krüger, I.: "Modeling Crosscutting Services with UML Sequence Diagrams". *Model Driven Engineering Languages and Systems*, Springer, LNCS 3713, pp. 522–536 (2005)
- [45] *Early Aspects website*. <http://www.early-aspects.net/> (accessed November 2010)
- [46] Elammari, M. and Lalonde, W.: "An Agent-Oriented Methodology: High-Level View and Intermediate Models". *1st International Workshop on Agent-Oriented Information Systems (AOIS)*, Heidelberg, Germany (June 1999)
- [47] France, R., Fleurey, F., Reddy, R., Baudry, B., and Ghosh, S.: "Providing Support for Model Composition in Metamodels". *11th IEEE International Enterprise Computing Conference (EDOC'07)*, Annapolis, Maryland, USA, IEEE Computer Society Press, pp. 253–266 (2007)
- [48] Ghanavati, S., Amyot D., and Peyton, L.: "Towards a Framework for Tracking Legal Compliance in Healthcare". *19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*, Trondheim, Norway, Springer, LNCS 4495, pp. 218–232 (June 2007)

- [49] Ghanavati, S.: *A Compliance Framework for Business Processes Based on URN*. M.Sc. thesis, SITE, University of Ottawa, Canada (2007)
- [50] Gil, A.: *Integrating Early Aspects with Goal-Oriented Requirements Engineering: The Case of KAOS*. Mestre em Engenharia Informática, Departamento de Informática, Universidade Nova de Lisboa, Portugal (2008)
- [51] Gross, D., and Yu, E.: “Dealing with System Qualities During Design and Composition of Aspects and Modules: An Agent and Goal-Oriented Approach”. *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Automated Software Engineering Conference*, Edinburgh, Scotland, UK, pp. 1–8 (October 2002)
- [52] Guan, R.: *From Requirements to Scenarios through Specifications: A Translation Procedure from Use Case Maps to LOTOS*. M.Sc. thesis, OCICS, University of Ottawa, Canada (2002)
- [53] Hamou-Lhadj, A., Braun, E., Amyot, D., and Lethbridge, T.: “Recovering Behavioral Design Models from Execution Traces”. *9th European Conference on Software Maintenance and Reengineering (CSMR)*, IEEE Computer Society Press, pp. 112–121 (March 2005)
- [54] Hassine, J., Rilling, J., and Dssouli, R.: “An ASM Operational Semantics for Use Case Maps”. *13th IEEE International Requirement Engineering Conference (RE’05)*, IEEE Computer Society Press, pp. 467–468 (September 2005)
- [55] Hassine, J., Rilling, J., and Dssouli, R.: “Formal Verification of Use Case Maps with Real Time Extensions”. *13th SDL Forum (SDL’07)*, Paris, France, Springer, LNCS 4745, pp. 225–241 (September 2007)
- [56] Hassine, J.: *Formal Semantics and Verification of Use Case Maps*. Ph.D. thesis, Concordia University, Montreal, Canada (April 2008)
- [57] Hodges, J. and Visser, J.: “Accelerating Wireless Intelligent Network Standards Through Formal Techniques”. *IEEE 1999 Vehicular Technology Conference (VTC’99)*, Houston, Texas, USA (1999)
- [58] ITU-T: *Message Sequence Charts (MSC), ITU-T Recommendation Z.120*. Geneva, Switzerland (April 2004); <http://www.itu.int/rec/T-REC-Z.120/en> (accessed November 2010)
- [59] ITU-T: *User Requirements Notation (URN) – Language Requirements and Framework, ITU-T Recommendation Z.150 (02/03)*. Geneva, Switzerland (February 2003); <http://www.itu.int/rec/T-REC-Z.150/en> (accessed November 2010), <http://www.UseCaseMaps.org/urn> (accessed November 2010)
- [60] ITU-T: *User Requirements Notation (URN) – Language definition, ITU-T Recommendation Z.151 (11/08)*. Geneva, Switzerland (November 2008); <http://www.itu.int/rec/T-REC-Z.151/en> (accessed November 2010)
- [61] Jacobson, I. and Ng, P.-W.: *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley (2005)

- [62] *jUCMNav website*. University of Ottawa; <http://jucmnav.softwareengineering.ca/jucmnav> (accessed November 2010)
- [63] Kaiya, H. and Saeki, M.: “Weaving Multiple Viewpoint Specifications in Goal-Oriented Requirements Analysis”. *11th Asia-Pacific Software Engineering Conference (APSEC 2004)*, Busan, Korea, IEEE Computer Society Press, pp. 418–427 (November 2004)
- [64] Kealey, J. and Amyot, D.: “Enhanced Use Case Map Traversal Semantics”. *13th SDL Forum (SDL'07)*, Paris, France, Springer, LNCS 4745, pp. 133–149 (September 2007)
- [65] Kealey, J., Kim, Y., Amyot, D., and Mussbacher, G.: “Integrating an Eclipse-Based Scenario Modeling Environment with a Requirements Management System”. *2006 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE06)*, Ottawa, Canada, pp. 2432–2435 (May 2006)
- [66] Kealey, J.: *Enhanced Use Case Map Analysis and Transformation Tooling*. M.Sc. thesis, OCICS, University of Ottawa, Canada (2007)
- [67] Kellens, A., Gybels, K., Brichau, J., and Mens, K.: “A Model-Driven Pointcut Language for More Robust Pointcuts”. *Workshop on Software Engineering Properties of Languages for Aspect Technology (SPLAT! 2006)*, Bonn, Germany (March 2006)
- [68] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J.: “Aspect-Oriented Programming”. *ECOOP'97 – Object Oriented Programming, 11th European Conference*, Springer, LNCS 1241, pp. 220–242 (June 1997)
- [69] Kienzle, J., Guelfi, N., and Mustafiz, S.: “Crisis Management Systems: A Case Study for Aspect-Oriented Modeling”. *Transactions on Aspect-Oriented Software Development (TAOSD) VII*, Springer, LNCS 6210, pp. 1–22 (2010)
- [70] Kiepuszewski, B., ter Hofstede, A.H.M., and van der Aalst, W.M.P.: “Fundamentals of control flow in workflows”. *Acta Informatica*, Springer, Vol. 39(3), pp. 143–209 (2003)
- [71] Kiepuszewski, B.: *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. Ph.D. thesis, QUT, Brisbane, Australia (2003)
- [72] Klein, J., Hérouet, L., and Jézéquel, J.-M.: “Semantic-based Weaving of Scenarios”. *5th International Conference on Aspect-Oriented Software Development (AOSD'06)*, Bonn, Germany, ACM, pp. 27–38 (March 2006)
- [73] Kohno, T., Stubblefield, A., Rubin, A., and Wallach, D.: “Analysis of an Electronic Voting System”. *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, pp. 27–40 (2004)
- [74] Koppen, C. and Stoerzer, M.: “Pcdiff: Attacking the Fragile Pointcut Problem”. *First European Interactive Workshop on Aspects in Software (EIWAS'04)*, Berlin, Germany (September 2004)

- [75] *LavaBlast Software Inc. website*. Canada; <http://www.lavablast.com> (accessed November 2010)
- [76] Lee, J., Wu, C.-L., Lee, W.-T., and Hsu, K.-H.: "Aspect-Enhanced Goal-Driven Sequence Diagram". *International Journal of Intelligent Systems (IJIS)*, Wiley, Vol. 25(8), pp. 712–732 (2010)
- [77] Leelaprute, P., Nakamura, M., Matsumoto, K., and Kikuno, T.: "Derivation and Evaluation of Feature Interaction Prone Scenarios with Use Case Maps". *IEICE Transactions on Communications*, Vol. J88-B(7), pp. 1237–1247 (July 2005)
- [78] Mehner, K., Monga, M., and Taentzer, G.: "Analysis of Aspect-Oriented Model Weaving". Rashid, A. and Ossher, H. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) V*, Springer, LNCS 5490, pp. 235–263 (2009)
- [79] Miga, A.: *Application of Use Case Maps to System Design with Tool Support*. M.Eng. thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada (October 1998)
- [80] Monteiro, C., Ramos, R., Castro, J., Araújo, J., Moreira, A., and Alencar, F.: "The iAspectPlugin to Automate the Identification of Crosscutting Concerns on i* Models". *1st Latin American Workshop on Aspect-Oriented Software Development (LA-WASP'07)*, João Pessoa, Brazil (2007)
- [81] Moreira, A. and Araújo, J.: "Handling Unanticipated Requirements Change with Aspects". *Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Banff, Canada (June 2004)
- [82] Moreira, A., Araújo, J., and Brito, I.: "Crosscutting Quality Attributes for Requirements Engineering". *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Ischia, Italy, ACM, pp. 167–174 (July 2002)
- [83] Morin, B., Klein, J., Barais, O., and Jézéquel, J.-M.: "A Generic Weaver for Supporting Product Lines". *Proceedings of the 13th International Workshop on Early Aspects*, Leipzig, Germany, ACM, pp. 11–18 (2008)
- [84] Mosser, S., Mussbacher, G., Blay-Fornarino, M., and Amyot, D.: "From Aspect-oriented Requirements Models to Aspect-oriented Business Process Design Models – An Iterative and Concern-Driven Approach for Software Engineering". *10th International Conference on Aspect-Oriented Software Development (AOSD 2011)*, Porto de Galinhas, Brazil (March 2011, to appear)
- [85] Mussbacher, G. and Amyot, D.: "Assessing the Applicability of Use Case Maps for Business Process and Workflow Description". *3rd International MCeTech Conference on eTechnologies*, Montreal, Canada, IEEE Computer Society Press, pp. 219–222 (January 23-25, 2008)
- [86] Mussbacher, G. and Amyot, D.: "Extending the User Requirements Notation with Aspect-oriented Concepts". *14th SDL Forum (SDL 2009)*, Bochum, Germany (September 2009). Reed, R., Bilgic, A., Gotzhein, R. (Editors), *SDL 2009: Design for Motes and Mobiles*, Springer, LNCS 5719, pp. 115–132 (2009)

- [87] Mussbacher, G. and Amyot, D.: “Heterogeneous Pointcut Expressions”. *Early Aspects Workshop at ICSE'09*, Vancouver, Canada, IEEE Computer Society Press, pp. 8–13 (May 18, 2009)
- [88] Mussbacher, G. and Amyot, D.: “On Modeling Interactions of Early Aspects with Goals”. *Early Aspects Workshop at ICSE'09*, Vancouver, Canada, IEEE Computer Society Press, pp. 14–19 (May 18, 2009)
- [89] Mussbacher, G., Amyot, D., and Weiss, M.: “Formalizing Patterns with the User Requirements Notation”. Taibi, T. (Editor), *Design Pattern Formalization Techniques*, IGI Global, pp. 304–325 (March 2007)
- [90] Mussbacher, G., Amyot, D., and Weiss, M.: “Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps”. *International Workshop on Requirements Engineering Visualization (REV 2006)*, Minneapolis, Minnesota, USA (September 2006)
- [91] Mussbacher, G., Amyot, D., and Weiss, M.: “Visualizing Early Aspects with Use Case Maps”. Rashid, A. and Aksit, M. (Editors), *Transactions on Aspect-Oriented Software Development III*, Springer, LNCS 4620, pp. 105–143 (November 2007)
- [92] Mussbacher, G., Amyot, D., and Whittle, J.: “Refactoring-Safe Modeling of Aspect-Oriented Scenarios”. *ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MODELS 2009)*, Denver, Colorado, USA (October 2009). Schürr, A. and Selic, B. (Editors), *Model Driven Engineering Languages and Systems*, Springer, LNCS 5795, pp. 286–300 (2009) (Acceptance rate: 18%)
- [93] Mussbacher, G., Amyot, D., and Whittle, J.: “Semantic-Based Aspect Interaction Detection with Goal Models (Position Paper)”. *10th International Conference on Feature Interactions (ICFI 2009)*, Lisbon, Portugal (June 11-12, 2009). Nakamura, M. and Reiff-Marganiec S. (Editors), *Feature Interactions in Software and Communication Systems X*, IOS Press, pp. 176–182 (2009)
- [94] Mussbacher, G., Amyot, D., Araújo, J., and Moreira, A.: “Modeling Software Product Lines with AoURN”. *Early Aspects Workshop: Early Aspects and Software Product Lines (EA-AOSD'08)*, Brussels, Belgium, ACM Digital Library (March 31, 2008)
- [95] Mussbacher, G., Amyot, D., Araújo, J., and Moreira, A.: “Requirements Modeling with the Aspect-oriented User Requirements Notation (AoURN): A Case Study”. Katz, S., Mezini, M., and Kienzle, J. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) VII*, Springer, LNCS 6210, pp. 23–68 (2010)
- [96] Mussbacher, G., Amyot, D., Araújo, J., Moreira, A., and Weiss, M.: “Visualizing Aspect-Oriented Goal Models with AoGRL”. *2nd International Workshop on Requirements Engineering Visualization (REV 2007)*, New Delhi, India (October 2007)
- [97] Mussbacher, G., Amyot, D., Weigert, T., and Cottenier, T.: “Feature Interactions in Aspect-Oriented Scenario Models”. *10th International Conference on Feature*

- Interactions (ICFI 2009)*, Lisbon, Portugal (June 11-12, 2009). Nakamura, M. and Reiff-Marganiec S. (Editors), *Feature Interactions in Software and Communication Systems X*, IOS Press, pp. 75–90 (2009)
- [98] Mussbacher, G., Amyot, D., Whittle, J., and Weiss, M.: “Flexible and Expressive Composition Rules with Aspect-oriented Use Case Maps (AoUCM)”. *10th International Workshop on Early Aspects (EA 2007)*, Vancouver, Canada (March 13, 2007). Moreira, A. and Grundy, J. (Editors), *Early Aspects: Current Challenges and Future Directions*, Springer, LNCS 4765, pp. 19–38 (December 2007)
- [99] Mussbacher, G., Barone, D. and Amyot, D.: “Towards a Taxonomy of Syntactic and Semantic Matching Mechanisms for Aspect-oriented Modeling”. *6th Workshop on System Analysis and Modelling (SAM 2010)*, Oslo, Norway (October 2010)
- [100] Mussbacher, G., Whittle, J., and Amyot, D.: “Modeling and Detecting Semantic-Based Interactions in Aspect-Oriented Scenarios”. *Requirements Engineering Journal (REJ)*, Springer, 15(2), pp. 197–214 (2010)
- [101] Mussbacher, G., Whittle, J., and Amyot, D.: “Semantic-Based Interaction Detection in Aspect-Oriented Scenarios”. *17th IEEE International Requirements Engineering Conference (RE’09)*, Atlanta, Georgia, USA, IEEE Computer Society Press, pp. 203–212 (September 2009) (Acceptance rate: 21%)
- [102] Mussbacher, G., Whittle, J., and Amyot, D.: “Towards Semantic-Based Aspect Interaction Detection”. *1st International Workshop on Non-functional System Properties in Domain Specific Modeling Languages (NFPinDSML2008)*, Toulouse, France (September 28, 2008)
- [103] Mussbacher, G.: “Aspect-Oriented User Requirements Notation”. Giese, H. (Editor), *Models in Software Engineering: Workshops and Symposia at MODELS 2007*, Springer, LNCS 5002, pp. 305–316 (August 2008)
- [104] Mussbacher, G.: “Evolving Use Case Maps as a Scenario and Workflow Description Language”. *10th Workshop on Requirements Engineering (WER’07)*, Toronto, Canada, pp. 56–67 (May 2007); http://www.inf.puc-rio.br/~wer/WERpapers/artigos/artigos_WER07/Hwer07-mussbacher.pdf (accessed November 2010)
- [105] Niu, N. and Easterbrook, S.: "Analysis of Early Aspects in Requirements Goal Models: A Concept-Driven Approach". Rashid, A. and Aksit, M. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) III*, Springer, LNCS 4620, pp. 40–72 (2007)
- [106] Niu, N. and Easterbrook S.: “Discovering Aspects in Requirements with Repertory Grid”. *Workshop on Early Aspects, co-located with ICSE 2006*, Shanghai, China (May 2006)
- [107] Niu, N., Yu, Y., González-Baixauli, B., Ernst, N., Sampaio do Prado Leite, J., and Mylopoulos, J.: "Aspects across Software Life Cycle: A Goal-Driven Approach". Katz, S., Ossher, H., France, R., and Jézéquel, J.-M. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) VI*, Springer, LNCS 5560, pp. 83–110 (2009)

- [108] Ölvingson, C., Hallberg, N., Timpka, T., and Lindqvist, K.: “Requirements Engineering for Inter-Organizational Health Information Systems with Functions for Spatial Analyses: Modeling a WHO Safe Community Applying Use Case Maps”. *Methods of Information in Medicine*, Schattauer GmbH, Vol. 41(4), pp. 299–304 (2002)
- [109] OMG: *Business Process Modeling Notation (BPMN) 1.0*; http://www.bpmn.org/Documents/OMG_Final_Adopted_BPMN_1-0_Spec_06-02-01.pdf (accessed November 2010)
- [110] OMG: *Business Process Modeling Notation (BPMN) website*. <http://www.bpmn.org> (accessed November 2010)
- [111] OMG: *Object Constraint Language Version 2.0*. <http://www.omg.org/spec/OCL/2.0/> (accessed November 2010)
- [112] OMG: *UML 2.0 Superstructure Specification. OMG Formal Specification*. formal/05-07-04 (July 2005); <http://www.omg.org/cgi-bin/doc?formal/05-07-04> (accessed November 2010)
- [113] OMG: *Unified Modeling Language (UML) website*. <http://www.uml.org> (accessed November 2010)
- [114] *OpenOME website*. University of Toronto; <http://www.cs.toronto.edu/km/opename/> (accessed November 2010)
- [115] Organization for the Advancement of Structured Information Standards (OASIS): *Web Services Business Process Execution Language (WSBPEL) Technical Committee (TC)*; http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (accessed November 2010)
- [116] Petriu, D.B. and Woodside, M.: “Software performance models from system scenarios.” *Performance Evaluation*, Elsevier, Vol. 61(1), pp. 65–89 (June 2005)
- [117] Petriu, D.B. and Woodside, M.: “Software Performance Models from System Scenarios in Use Case Maps”. Field, T., Harrison, P.G., Bradley, J., and Harder, U. (Editors), *Computer Performance Evaluation*, Springer, LNCS 2324, pp. 141–158 (April 2002)
- [118] Pourshahid, A., Chen, P., Amyot, D., Forster, A.J., Ghanavati, S., Peyton, L., and Weiss, M.: “Toward an integrated User Requirements Notation framework and tool for Business Process Management”. *3rd International MCE Tech Conference on eTechnologies*, Montreal, Canada, IEEE Computer Society Press, pp. 3–15 (January 2008)
- [119] Pourshahid, A., Chen, P., Amyot, D., Weiss, M., and Forster, A.: “Business Process Monitoring and Alignment: An Approach Based on the User Requirements Notation and Business Intelligence Tools”. *10th Workshop on Requirements Engineering*, Toronto, Canada, pp. 149–159 (May 2007); http://www.inf.puc-rio.br/~wer/WERpapers/artigos/artigos_WER07/Jwer07-amyot.pdf (accessed November 2010)

- [120] Pourshahid, A., Mussbacher, G., Amyot, D., and Weiss, M.: “An Aspect-Oriented Framework for Business Process Improvement”. *4th International MCEtech Conference on eTechnologies*, Ottawa, Canada (May 4-6, 2009). Babin, G., Kropf, P., and Weiss, M. (Editors), *E-Technologies: Innovation in an Open World*, Springer, LNBIP 26, pp. 290–305 (2009)
- [121] Pourshahid, A., Mussbacher, G., Amyot, D., and Weiss, M.: “Toward an Aspect-Oriented Framework for Business Process Improvement”. *International Journal of Electronic Business (IJEB)*, Inderscience Publishers, 8(3), pp. 233–259 (2010)
- [122] Rashid, A., Moreira, A., and Araújo, J.: “Modularisation and Composition of Aspectual Requirements”. *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, Boston, Massachusetts, USA, ACM, pp. 11–20 (March 2003)
- [123] Roy, J.-F., Kealey, J., and Amyot, D.: “Towards Integrated Tool Support for the User Requirements Notation”. *SAM 2006: Language Profiles - 5th Workshop on System Analysis and Modelling*, Kaiserslautern, Germany, Springer, LNCS 4320, pp. 183–197 (May 2006)
- [124] Roy, J-F.: *Requirements Engineering with URN: Integrating Goals and Scenarios*. M.Sc. thesis, OCICS, University of Ottawa, Canada (March 2007)
- [125] Russell, N., ter Hofstede, A.H.M., Edmond, D., and van der Aalst, W.M.P.: “Workflow Data Patterns”. *QUT Technical report*, FIT-TR-2004-01, Queensland University of Technology, Brisbane, Australia (2004); http://workflowpatterns.com/documentation/documents/data_patterns BETA TR.pdf (accessed November 2010)
- [126] Russell, N., ter Hofstede, A.H.M., Edmond, D., and van der Aalst, W.M.P.: “Workflow Resource Patterns”. *BETA Working Paper Series*, WP 127, Eindhoven University of Technology, Eindhoven, Netherlands (2004); [http://workflowpatterns.com/documentation/documents/Resource Patterns](http://workflowpatterns.com/documentation/documents/Resource%20Patterns) BETA TR.pdf (accessed November 2010)
- [127] Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., and Mulyar, N.: “Workflow Control-Flow Patterns: A Revised View”. *BPM Center Report BPM-06-22*, BPMcenter.org (2006); <http://workflowpatterns.com/documentation/documents/BPM-06-22.pdf> (accessed November 2010)
- [128] Russell, N., van der Aalst, W.M.P., and ter Hofstede, A.H.M.: “Exception Handling Patterns in Process-Aware Information Systems”. *BPM Center Report*, BPM-06-04, BPMcenter.org (2006); <http://workflowpatterns.com/documentation/documents/BPM-06-04.pdf> (accessed November 2010)
- [129] Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., and Wohed, P.: “On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling”. *Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006)*, Hobart, Australia; Stumptner, M., Hartmann, S., and Kiyoki, Y. (Editors), CRPIT, Vol. 53, pp. 95–104 (2006)

- [130] Sampaio, A., Chitchyan, R., Rashid, A., and Rayson, P.: "EAMiner: a Tool for Automating Aspect-Oriented Requirements Identification". *20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Long Beach, California, USA, ACM, pp. 352–355 (2005)
- [131] Sampaio, A., Rashid, A., Chitchyan, R., and Rayson, P.: "EA-Miner: Towards Automation in Aspect-Oriented Requirements Engineering". Rashid, A. and Ak-sit, M. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) III*, Springer, LNCS 4620, pp. 4–39 (2007)
- [132] *Sandrila SDL website*. SanDriLa Ltd; <http://www.sandrila.co.uk/visio-sdl/index.php> (accessed November 2010)
- [133] Sant'Anna, C. *et al.*: "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework". *Brazilian Symposium on Software Engineering (SBES'03)*, Manaus, Brazil, pp. 19–34 (October 2003)
- [134] Schauerhuber, A., Schwinger, W., Kapsammer, E., Retschitzegger, W., Wimmer, M., and Kappel, G.: "A Survey on Aspect-Oriented Modeling Approaches". *Technical Report*, Vienna University of Technology, Austria (October 2007)
- [135] Scratchley, W.C. and Woodside, C.M.: "Evaluating Concurrency Options in Software Specifications". *7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, College Park, Maryland, USA, pp. 330–338 (October 1999)
- [136] Shiri, M., Hassine, J., and Rilling, J.: "Feature Interaction Analysis: A Maintenance Perspective". *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Atlanta, Georgia, USA, pp. 437–440 (November 2007)
- [137] Siddiqui, K.H. and Woodside, C.M.: "Performance aware software development (PASD) using resource demand budgets". *Workshop on Software and Performance (WOSP)*, Rome, Italy, pp. 275–285 (July 2002)
- [138] Silva, L., Batista, T., Garcia, A., Medeiros, A., and Minora, L.: "On the Symbiosis of Aspect-Oriented Requirements and Architectural Descriptions". *10th International Workshop on Early Aspects (EA 2007)*, Vancouver, Canada (March 13, 2007). Moreira, A. and Grundy, J. (Editors), *Early Aspects: Current Challenges and Future Directions*, Springer, LNCS 4765, pp. 75–93 (December 2007)
- [139] Sousa G., Soares, S., Borba P., and Castro J.: "Separation of Crosscutting Concerns from Requirements to Design: Adapting an Use Case Driven Approach", *Workshop on Early Aspects*, Lancaster, England, UK (March 2004)
- [140] Strembeck, M. and Zdun, U.: "Modeling Interdependent Concern Behavior Using Extended Activity Models". *Journal of Object Technology*, Vol. 7(6), pp. 143–166 (July-August 2008)
- [141] Tarr, P., Ossher, H., Harrison, W., and Sutton, S.M.: "N degrees of separation: Multidimensional separation of concerns". *Proceedings of the 21st International Conference on Software Engineering (ICSE 99)*, IEEE, Los Angeles, ACM, pp. 107–119 (May 1999)

- [142] *UCMNav website*. <http://jucmnav.softwareengineering.ca/twiki/bin/view/UCM/UcmNav> (accessed November 2010)
- [143] *URN Metamodel for jUCMNav tool*. <http://jucmnav.softwareengineering.ca/twiki/bin/view/ProjetSEG/URNMetaModel> (accessed November 2010)
- [144] *URN Virtual Library*. <http://www.usecasemaps.org/pub> (accessed November 2010)
- [145] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., and Barros, A.P.: “Workflow Patterns”. *Distributed and Parallel Databases*, Vol. 14(3), pp. 5–51 (July 2003)
- [146] van Lamsweerde, A.: *Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley & Sons Ltd (2009)
- [147] Weiss, M. and Amyot, D.: “Business Process Modeling with URN”. *International Journal of E-Business Research*, Vol. 1(3), pp. 63–90 (July-September 2005)
- [148] Weiss, M., Esfandiari, B., and Luo, Y.: “Towards a Classification of Web Service Feature Interactions”. *Computer Networks*, Elsevier, Vol. 51(2), pp. 359–381 (February 2007)
- [149] Weiss, M., Esfandiari, B., and Luo, Y.: “Towards a Classification of Web Service Feature Interactions”. *International Conference on Service-Oriented Computing (ICSOC)*, Amsterdam, Netherlands, Springer, LNCS 3826, pp. 101–114 (November 2005)
- [150] White, S.A.: “Process Modeling Notations and Workflow Patterns”. Fischer, L. (Editor), *Workflow Handbook 2004*, Future Strategies Inc., pp. 265–294 (2004); http://www.bpmn.org/Documents/Notations_and_Workflow_Patterns.pdf (accessed November 2010)
- [151] Whittle, J. and Araújo, J.: “Scenario Modelling with Aspects”. *IEE Proceedings – Software*, Vol. 151(4), pp. 157–172 (August 2004)
- [152] Whittle, J. and Jayaraman, P.: “MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation”, Giese, H. (Editor), *MoDELS 2007 Workshops*, Springer, LNCS 5002, pp. 16–27 (2008)
- [153] Whittle, J., Araújo, J., Moreira, A., and Rabbi, R.: “Graphical Composition of State-Dependent Use Case Behavioral Models”. *ISE Department Technical Report*, George Mason University, Fairfax, Virginia, USA, ISE-TR-07-01, <http://cs.gmu.edu/~tr-admin/papers/ISE-TR-07-01.pdf> (accessed November 2010)
- [154] Whittle, J., Jayaraman, P., Elkhodary, A., Moreira, A., and Araújo, J.: “MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation”. Katz, S., Ossher, H., France, R., and Jézéquel, J.-M. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) VI*, Springer, 5560, pp. 191–237 (2009)
- [155] Whittle, J., Moreira, A., Araújo, J., Jayaraman, P., Elkhodary, A., and Rabbi, R.: “An Expressive Aspect Composition Language for UML State Diagrams”. *Model*

- Driven Engineering Languages and Systems, 10th International Conference, MODELS 2007*, Springer, LNCS 4735, pp. 514–528 (2007)
- [156] Wohed, P., van der Aalst, W.M.P., Dumas, M. and ter Hofstede, A.H.M.: “Analysis of Web Services Composition Languages: The Case of BPEL4WS”. *Proceedings 22nd International Conference on Conceptual Modelling (ER)*, Chicago, Illinois, USA, October 13-16, pp. 200–215 (2003)
- [157] Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., and Russell, N.: “Pattern-based Analysis of UML Activity Diagrams”. *BETA Working Paper Series*, WP 129, Eindhoven University of Technology, Eindhoven, Netherlands (2004); http://is.tm.tue.nl/research/patterns/download/uml2patterns_BETA_TR.pdf (accessed November 2010)
- [158] *Workflow Patterns website*. <http://www.workflowpatterns.com> (accessed November 2010)
- [159] *YAWL: Yet Another Workflow Language website*. <http://yawlfoundation.org> (accessed November 2010)
- [160] Yu, E.: *Modeling Strategic Relationships for Process Reengineering*. Ph.D. thesis, Department of Computer Science, University of Toronto, Canada (1995)
- [161] Yu, Y., Sampaio do Prado Leite, J., and Mylopoulos, J.: “From Goals to Aspects: Discovering Aspects from Requirements Goal Models”. *12th International Requirements Engineering Conference (RE'04)*, Kyoto, Japan (September 2004)
- [162] Zduin, U. and Strembeck, M.: “Modeling the Evolution of Aspect Configurations using Model Transformations”. *Proceedings of the Linking Aspect Technology and Evolution Workshop (LATE)*, Bonn, Germany (March 2006)
- [163] Zhang, J., Cottenier, T., van den Berg, A., and Gray, J.: “Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver”. *Journal of Object Technology, Special Issue on Aspect-Oriented Modeling*, Vol. 6(7), pp. 89–108 (August 2007)
- [164] Zhang, J., Liu, Y., Jiang, M., and Strassner, J.: “An Aspect-Oriented Approach to Handling Crosscutting Concerns in Activity Modeling”. *International MultiConference of Engineers and Computer Scientists (IMECS) 2008*, Hong Kong, pp. 885–890 (March 2008)

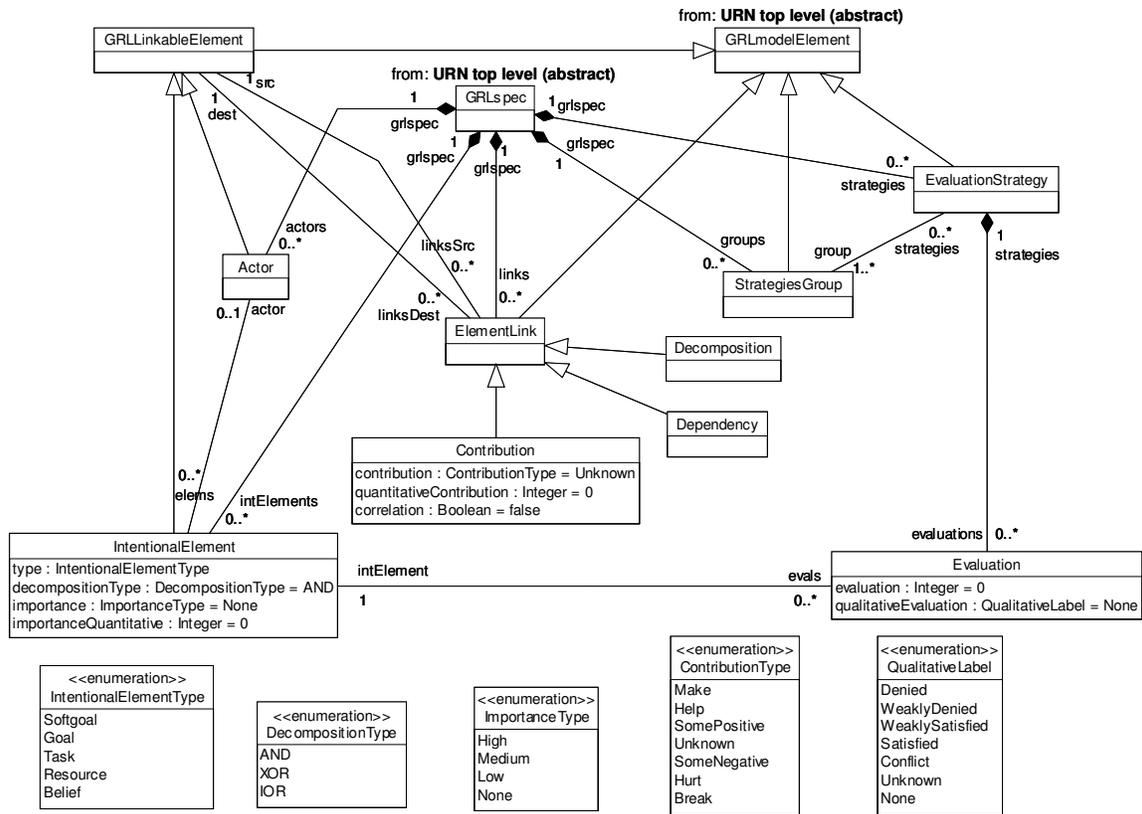


Figure 162 Abstract Syntax: GRL

Figure 162 presents the complete metamodel of the abstract syntax of GRL. The diagram shows a) the relationship of GRLLinkableElements with ElementLinks on the left hand side and in the middle and b) the concepts required for the evaluation of GRL models (StrategiesGroup, EvaluationStrategy, and Evaluation) on the right hand side.

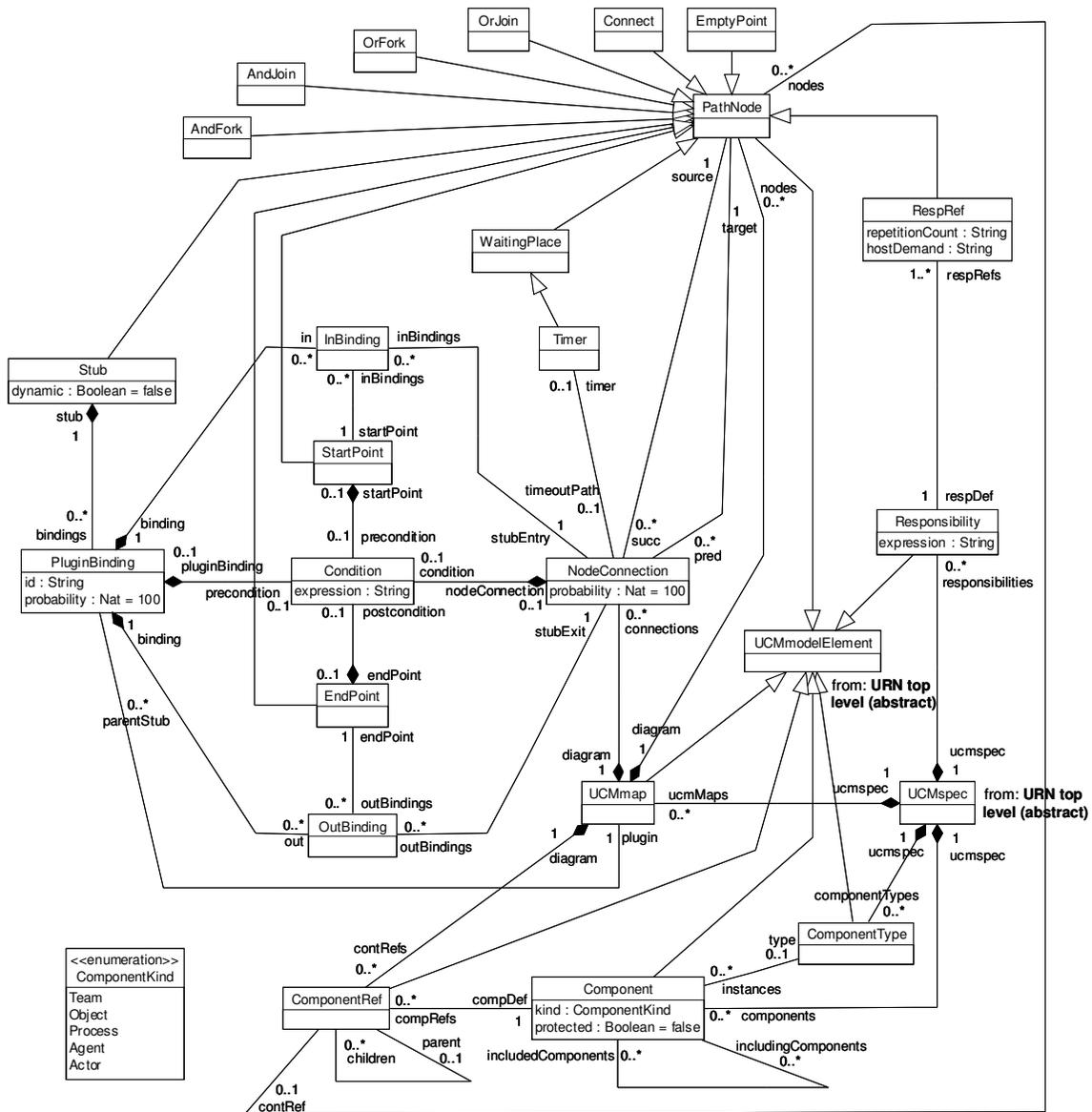


Figure 163 Abstract Syntax: UCM Core Overview

Figure 163 presents the core metamodel of the abstract syntax of the UCM notation. The diagram roughly shows a) path-related concepts at the top, b) plug-in binding-related concepts on the left middle, and c) component-related concepts at the bottom of the figure.

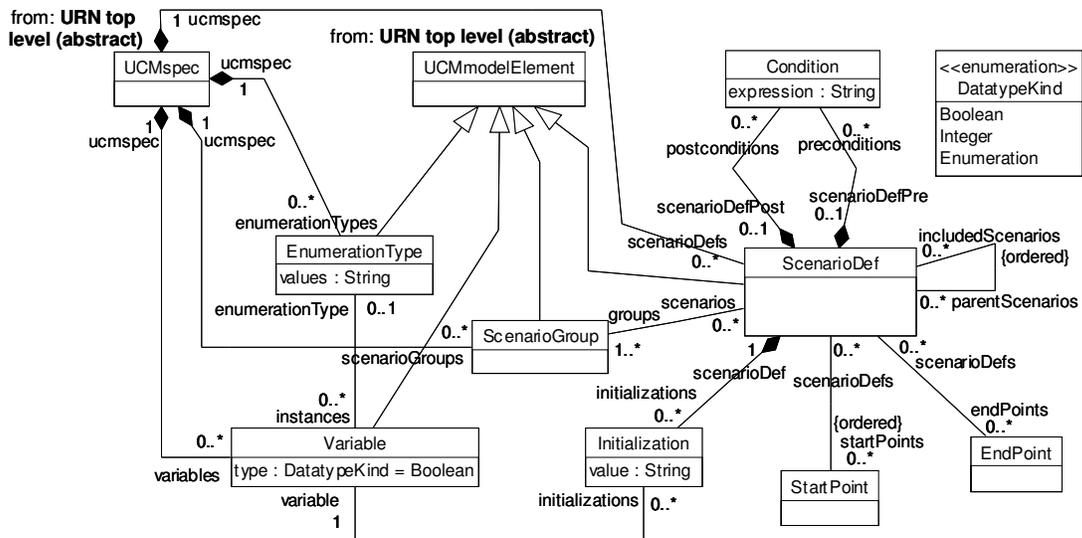


Figure 164 Abstract Syntax: UCM Scenarios Overview

Figure 164 presents the metamodel of the abstract syntax of UCM scenarios. The diagram also introduces further relationships for StartPoint and EndPoint.

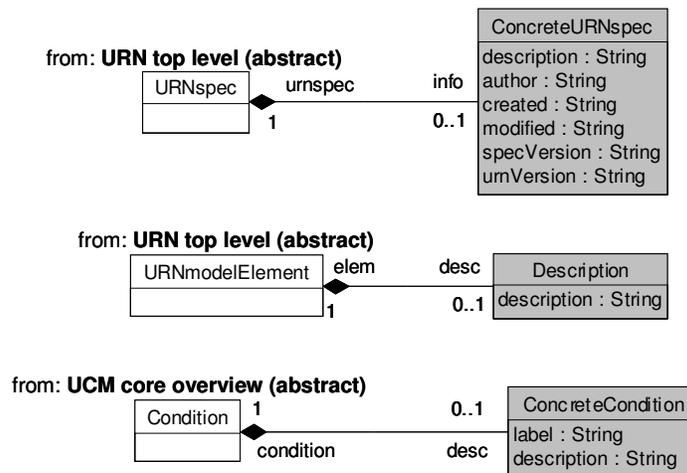


Figure 165 Concrete Syntax: URN Top Level

Figure 165 shows the top level of the metamodel of the concrete syntax of URN, which extends the metamodel of the abstract syntax in Figure 161. The diagram shows all metamodel classes of the concrete syntax in grey color.

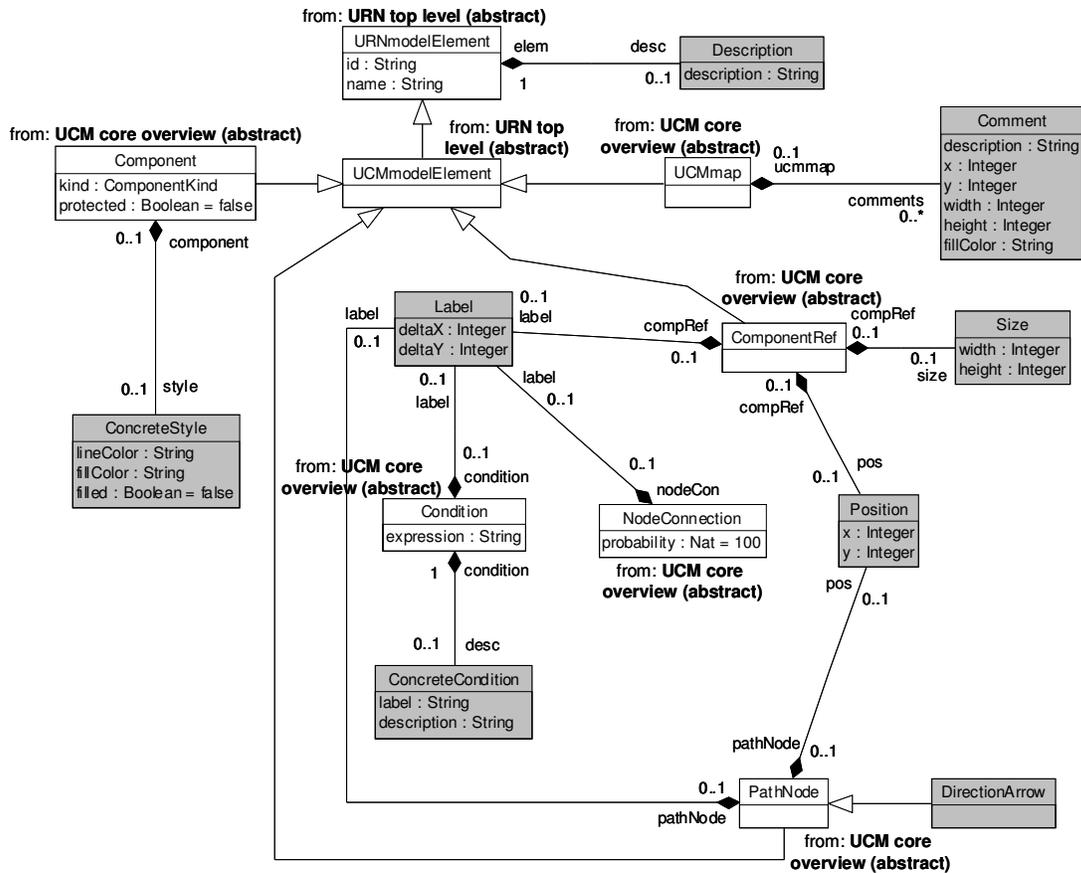


Figure 167 Concrete Syntax: UCM

Figure 167 presents the metamodel of the concrete syntax of the UCM notation, which extends the metamodel of the abstract syntax in Figure 161, Figure 163, and Figure 164. The diagram shows all metamodel classes of the concrete syntax in grey color.

Appendix B: UCM 2.0 Traversal Mechanism

Table 15 lists all requirements for the improved traversal mechanism based on the findings of Chapter 3, Chapter 4, and Chapter 6 whereas Table 16 lists those of the improved scenario definitions. Table 17 defines a glossary of terms used in the requirements statements. These terms are underlined and shown in bold in the requirements. Note that the requirements for a type of stub apply to all subtypes (see Figure 94) unless more specific requirements are stated for the subtype.

The list of requirements in Table 15 is not a complete list of requirements for the UCM path traversal mechanism but needs to be complemented by the set of requirements that describes the basic behavior of the path traversal mechanism and is already implemented in the jUCMNav tool. A complete list of requirements can be found in the latest version of the URN standard [60]. Many requirements from Table 15 are incorporated into that version as indicated in parentheses by the ID from the standard. The requirements which are not incorporated at this point are indicated by an X in parentheses. Requirements that are incorporated as a constraint are indicated by a C in parentheses.

Table 15 Requirements for the UCM 2.0 Traversal Mechanism

ID	Requirement for UCM 2.0 Traversal Mechanism (TM)	Source of Requirement
1a (52)	Upon arrival at a singleton map M, the TM shall traverse the only instance of M that exists in the UCM model.	Singleton Maps
1b (53)	Upon arrival at an unconnected start point S of a non-singleton traversal root map M, the TM shall traverse the N th instance of M where N is the number of times S has been visited in the current traversal.	
1c (54)	Upon arrival at a non-singleton plug-in map M of a non-synchronizing stub S, the TM shall traverse a) a different instance of M per different instance of S and b) the same instance of M for the same instance of S.	
1d (55)	Upon arrival at a non-singleton replicated plug-in map M of a non-synchronizing stub S, the TM shall traverse a) a different set of replicated instances of M per different instance of S and b) the same set of replicated instances of M for the same instance of S.	

ID	Requirement for UCM 2.0 Traversal Mechanism (TM)	Source of Requirement
1e (56)	Upon arrival at a non-singleton plug-in map M along an in-path of a synchronizing stub S, the TM shall traverse a) a different instance of M per different instance of a stub, b) the N th instance of M for this instance of S during the N th visit , and c) the same instance of M for the same instance of S in the same visit .	
1f (57)	Upon arrival at a non-singleton replicated plug-in map M along an in-path of a synchronizing stub S, the TM shall traverse a) a different set of replicated instances of M per different instance of a stub, b) the N th set of replicated instances of M for this instance of stubs during the N th visit , and c) the same set of replicated instances of M for the same instance of S in the same visit .	
1g (C)	The TM shall issue a warning and ignore the replication factor, if a singleton map has a replication factor greater than one.	
2a (X)	Upon arrival at an OR-fork O without conditions, the TM shall traverse the branch of O specified in the scenario definition.	WCP-16 Deferred Choice WCP-18 Milestone
2b (X)	Upon arrival at a dynamic stub S without a selection policy, the TM shall traverse the plug-in map of S specified in the scenario definition.	
3a (40,42,43)	Upon arrival at a dynamic stub S, the TM shall traverse in parallel all the plug-in map instances of S with conditions evaluating to true.	WCP-06 Multi-Choice
3b (10,40,42,43)	Upon arrival at a dynamic stub S, the TM shall stop the traversal of the path with a warning, if all conditions of the plug-in map instances of S evaluate to false.	
4a (47)	The TM shall synchronize a synchronizing stub's plug-in map instances, only if they belong to the same visit .	WCP-07 Structured Syn. Merge WCP-09 Struct. Discriminator
4b (48)	Once for each visit upon first arrival at a synchronizing stub S with the default synchronization threshold for an out-path O, the TM shall set the synchronization threshold of O to the number of plug-in map instances of S with conditions evaluating to true.	WCP-13 MI with a priori DTK WCP-14 MI with a priori RTK WCP-26 Cancel MI Activity WCP-27 Complete MI Activity WCP-28 Blocking Discr.
4c (46)	The TM shall continue the traversal along an out-path O of a synchronizing stub S, when O has been reached by the plug-in map instances of S during the same visit as often as specified by the synchronization threshold of O.	WCP-29 Cancel. Discriminator WCP-30 Structured Partial Join WCP-31 Blocking Partial Join WCP-32 Cancelling Partial Join
4d (49)	The TM shall ignore the arrival of plug-in map instances at an out-path O of a synchronizing stub during a visit , if the synchronization threshold of O has been reached for the visit .	WCP-34 Static Part. Join for MI WCP-35 Canc. Part. Join for MI
4e (41)	Upon arrival at a dynamic stub S, the TM shall traverse in parallel the number of instances of a plug-in map M of S as specified by the replication factor of the plug-in binding of M.	
4f (50)	Upon arrival at a synchronizing stub S with blocking enabled, the TM shall allow an in-path of S to be traversed another time when all plug-in map instances of S have been traversed.	

ID	Requirement for UCM 2.0 Traversal Mechanism (TM)	Source of Requirement
4g (51)	When all plug-in map instances of a synchronizing stub S have been traversed in the N th visit , the TM shall treat an in-path of S as having been visited N times, if the in-path was visited less than N times.	
4h (X)	The TM shall stop the traversal of the plug-in map instances of a synchronizing stub S with cancelling enabled for the N th visit , when the synchronization thresholds for the out-paths of S have been reached for the N th visit .	
5 (58)	Upon entering a protected component reference C along a path P, the TM shall start the traversal of P when no other path is being traversed in any component reference of the component definition of C.	WCP-39 Critical Section
6 (59)	The TM shall interleave path nodes of parallel branches that are bound to the same component reference C, if the component definition of C is of kind object.	WCP-17 Interl. Parallel Routing WCP-40 Interleaved Routing
7a (X)	Upon arrival at a failure point F, the TM shall stop the traversal of the path and set the failure variable of F to true, if the failure condition of F evaluates to true.	WCP-19 Cancel Task WCP-20 Cancel Case WCP-25 Cancel Region
7b (X)	Upon arrival at a failure point F, the TM shall continue with the traversal past F, if the failure condition of F evaluates to false.	WCP-26 Cancel MI Activity WCP-27 Complete MI Activity
7c (X)	Upon arrival at a path node other than a stub where a failure occurs according to the scenario definition, the TM shall stop the traversal of the path and set the failure variable specified in the scenario definition to true.	
7d (X)	Upon arrival at a map, stub, component reference, component definition, or responsibility definition where a failure occurs according to the scenario definition, the TM shall a) duplicate the scenario for each path node where the failure could occur, b) stop the duplicated scenario at a different path node for each scenario, and c) set the failure variable specified in the scenario definition to true.	
7e (X)	Upon setting a failure variable V to true, the TM shall continue in parallel at the failure and abort start points that have a condition evaluating to true and then set the V to false.	
7f (X)	Upon continuing the traversal past an abort start point along an abort path, the TM shall stop the traversal on all map instances in the abort scope of the path.	
8a (29)	Upon arrival at a persistent waiting place, transient waiting place, persistent timer, or transient timer along the waiting path WP, the TM shall increase the number of arrived WPs by 1 (the initial number of arrived WPs is 0).	WCP-23 Transient Trigger WCP-24 Persistent Trigger
8b (30)	Upon arrival at a persistent waiting place or persistent timer along the trigger or release path TRP, the TM shall increase the number of arrived TRPs by 1 (the initial number of arrived TRPs is 0).	

ID	Requirement for UCM 2.0 Traversal Mechanism (TM)	Source of Requirement
8c (31)	Upon arrival at a transient waiting place or transient timer along the trigger or release path TRP, the TM shall set the number of arrived TRPs to 1, if the number of arrived waiting paths is greater than 0 (the initial number of arrived TRPs is 0).	
8d (32)	The TM shall continue the traversal past a waiting place WP when a) WP's condition evaluates to true or b) at least one waiting path and one trigger path have arrived at WP.	
8e (34, 35)	The TM shall continue the traversal past a timer T along its regular path RP when a) the condition of RP evaluates to true or b) the condition of RP evaluates to false, the condition of T's timeout path evaluates to false, and at least one waiting path and one release path have arrived at T.	
8d (36)	The TM shall continue the traversal past a timer T along its timeout path when the traversal cannot continue along T's regular path.	
8e (37, 38)	The TM shall decrease the number of arrived waiting paths NWP by 1 and the number of arrived trigger or release paths NTRP by 1 when continuing past the waiting place or timer unless the NWP or NTRP, respectively, is already 0.	
9a (60)	Upon arrival at a component reference C1 on a plug-in map instance with a component plug-in binding to a component reference C2 on the parent map instance, the TM shall use the component definition of C2 as the component definition of C1.	Component and Responsibility Plug-in Bindings
9b (61)	Upon arrival at a component reference C for which a plug-in binding is expected to be specified but is not, the TM shall issue a warning and continue with the traversal without replacing C.	
9c (X)	Upon arrival at a responsibility reference R1 on a plug-in map instance with a responsibility plug-in binding to a responsibility definition R2, the TM shall use R2 as the responsibility definition of R1.	
9d (X)	Upon arrival at a responsibility reference R for which a plug-in binding is expected to be specified but is not, the TM shall issue a warning and continue with the traversal without replacing R.	
10a (X)	Upon arrival at a local start point S, the TM shall stop the traversal of the path, if the map of S is not already traversed by the TM.	Local Start and End Points
10b (X)	The TM shall issue a warning and continue the traversal, if a local start point or local end point is used in a plug-in binding.	
11a (X)	Upon arrival at a dynamic aspect marker A, the TM shall traverse plug-in map instances of A in random order.	Aspect-oriented Modeling
11b (X)	Upon arrival at an aspect marker, the TM shall continue the traversal on the aspect map instance M at the start point or out-path of a pointcut stub as specified by the plug-in binding of M.	

ID	Requirement for UCM 2.0 Traversal Mechanism (TM)	Source of Requirement
11c (X)	Upon arrival at a pointcut stub, the TM shall continue the traversal at the out-path of an aspect marker, if such a plug-in binding is specified.	
11d (X)	Upon arrival at a pointcut stub, the TM shall stop the traversal of the path, if no plug-in binding to the out-path of an aspect marker is specified.	
11e (X)	Upon arrival at an aspect map instance's variable V from an aspect marker A, the TM shall substitute V with the path element matched against V as specified by the mapping for A.	
11f (X)	The TM shall allow the traversal to enter an aspect map instance M through an aspect marker A1 and exit M through another aspect marker A2, if A1 and A2 belong to the same aspect marker group.	
11g (X)	The TM shall ignore an aspect marker A if a corresponding aspect marker B has not been visited.	
11h (X)	The TM shall take lost hierarchies into account when traversing between map hierarchy levels.	

Table 16 Requirements for UCM 2.0 Scenario Definitions

ID	Requirement for UCM 2.0 Scenario Definitions (SD)	Source of Requirement
12a (X)	The SD shall allow the requirements engineer to select one of the out-going branches of an OR-fork without conditions to indicate continuation of the scenario.	WCP-16 Deferred Choice WCP-18 Milestone
12b (X)	The SD shall allow the requirements engineer to select one of the plug-in maps of a dynamic stub without a selection policy to indicate continuation of the scenario.	
13a (X)	The SD shall allow the requirements engineer to specify which failure occurs in the scenario by selecting a failure variable.	WCP-19 Cancel Task WCP-20 Cancel Case WCP-25 Cancel Region
13b (X)	The SD shall allow the requirements engineer to select a path node to indicate where the failure occurs in the scenario.	WCP-26 Cancel MI Activity WCP-27 Complete MI Activity
13c (X)	The SD shall allow the requirements engineer to select a map to indicate where the failure occurs in the scenario.	
13d (X)	The SD shall allow the requirements engineer to select a component reference to indicate where the failure occurs in the scenario.	
13e (X)	The SD shall allow the requirements engineer to select a component definition to indicate where the failure occurs in the scenario.	
13f (X)	The SD shall allow the requirements engineer to select a responsibility definition to indicate where the failure occurs in the scenario.	

Table 17 Glossary of Terms Used in Requirements

Term	Description
Abort Scope	The map instance where the abort path is specified plus all reachable paths on all map instances below that map instance in the map hierarchy established by the traversal mechanism for a scenario definition.
Traversal Root Map	A map that is at the highest level in the map hierarchy established by the traversal mechanism for a scenario definition.
Unconnected Start Point	A start point that is not directly connected to another end point or to another path.
Visit	A visit of a synchronizing stub is characterized by how often an in-path of the stub has been traversed. If an in-path is traversed the first time, then it is the first visit of the stub. If the same in-path is traversed the n th time, then it is the n th visit of the stub. If another in-path of the stub is traversed for the first time, then it is the first visit of the stub. Plug-in maps that have been instantiated because of a visit are said to belong to the visit.

Appendix C: BNF for Name Expressions

The grammar of expressions for names of responsibilities and components in pointcut expressions is defined by *<expression>* in the following BNF. Furthermore, conditions for aspect markers use *<condition>*. Responsibilities and components in aspect maps may use *<variable name>* in addition to regular names. The expressions for names of actors and intentional elements in pointcut expressions are defined by *<disjunction>*.

```
// An expression is either an assignment to a variable, or a disjunction
<expression> ::= <assignment> | <condition> | <disjunction>
<assignment> ::= <variable name> <assignment symb> <disjunction>
<condition> ::= <variable name> <condition symb> <disjunction>
<variable name> ::= <variable symb> <identifier>
<variable symb> ::= $
<assignment symb> ::= =
<condition symb> ::= ==

// A disjunction combines terms with and/or/xor/not and parentheses
<disjunction> ::= <conjunction> {{<or> | <xor>} <conjunction>}*
<conjunction> ::= <term unit> {<and> <term unit>}*
<term unit> ::= <negation> | <term>
<negation> ::= <not> <term unit>
<term> ::= { <leftparenthesis> <disjunction> <rightparenthesis>
            | <name> | <wildcarded name>
            | <variable name> }

<and> ::= &&
<or> ::= ||
<xor> ::= ^
<not> ::= !
<leftparenthesis> ::= (
<rightparenthesis> ::= )

// Wildcards can appear before, inside, or at the end of names,
// but two successive wildcard symbols are not allowed.
<wildcarded name> ::= { <wildcard>[<name><wildcard>]*
                        | <name>[<wildcard><name>]+
                        | <wildcard><name>
                        | <name><wildcard> }

<wildcard> ::= *

// Names must start with a letter or an underscore, and may contain spaces
// (but will not end with a space)
<name> ::= <identifier>[<space><word>]+
<identifier> ::= {<letter> | <underscore>} [<word>]
<word> ::= {<letter> | <underscore> | <decimal digit>}+
<letter> ::= (any printable alphabetical character defined in UCS
             [ITU-T T.55])
<decimal digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<underscore> ::= _
<space> ::= ␣
```

Appendix D: Matching Algorithm

A high level summary of the matching algorithm follows, showing how one pointcut expression is matched against the URN model. It makes use of the terms Match and Mapping as defined in Figure 111. In addition to the three operations `matchPointcutExpression`, `matchElement`, and `matchNeighbors` shown below, several other operations are required. More detailed descriptions of these operations can be found in the comments below. In general, the operation `postprocessAnything` deals with special cases for the anything pointcut element. It is used by `matchPointcutExpression`. The operation `matchPermutations` finds, given two sets of model elements, all permutations that match elements in one set against elements in the other. It makes use of `matchElement` to match individual elements and is used by `matchNeighbors`. Note that a) each element in the set of neighboring pointcut elements has to be matched against exactly one element in the set of neighboring base elements and b) each element in the set of neighboring base elements has to be matched against zero or one element in the set of neighboring pointcut elements. All other operations are related to the matching of one pointcut element against one base element and are used by `matchElement`.

```
Operation: matchPointcutExpression
Input: URNPointcutDiagram pcDiagram, URNmodel base
Output: MatchList
Exception: NoMatchFound

begin // matchPointcutExpression
MatchList result = ∅
URNmodelElement initPcElement = pcDiagram.getInitialElement()
foreach IntentionalElement, Actor, ElementLink, or PathNode el in base
String condition = ""
if matchElement(initPcElement, el, condition) then
MatchList firstMapping = ∅
add new Mapping(initPcElement, el, condition) to firstMapping
try
// Match neighboring elements of initPcElement against neighboring
// elements of el and return matches (which contains firstMapping
// plus new mappings found by matchNeighbors() including all
// permutations).
MatchList matches = matchNeighbors(initPcElement, el, firstMapping)
add matches to result
endtry
catch NoMatchFound
```

```

        // Do nothing and continue for loop (this gives initPcElement a
        // chance to be matched against other model elements in the base).
    endcatch
endf
endif
endforeach // IntentionalElement, Actor, ElementLink, or PathNode
// The following post-processing step deals with two situations for matches
// involving the anything pointcut element. 1) All matches with an anything
// pointcut element that fully contain another match with the same anything
// pointcut element are removed from the result.
// 2) Failed matches with an anything pointcut element that have been saved
// to a global variable by matchNeighbors() are restarted as described in
// Figure 112 and any successful matches are added to the result.
result = postprocessAnything(result)
if result == Ø then
    throw new NoMatchFound() // No mapping found at all.
endif
return result
end // matchPointcutExpression

```

```

Operation:    matchElement
Input:       URNmodelElement pcElement, URNmodelElement baseElement,
                String condition
Output:      Boolean
Exception:   ---

```

```

begin // matchElement
// Decides whether two model elements match according to the criteria
// mentioned in Section 7.1, taking conditional matches due to variables
// and component/ responsibility plug-in bindings into account.
// The anything pointcut element may be matched to any path node and path
// node connection in the base model.
if isAnythingMatch(pcElement) then
    return true;
endif
// Unnamed start and end points in the pointcut expression may be matched
// to any type of path node in the base model. All other criteria do not
// apply to unnamed start and end points. They are considered free matches.
if isUnnamedStartEndPointMatch(pcElement) then
    return true;
endif
// The types of the model elements must match. The following exceptions
// apply:
// A static stub in the pointcut expression may be matched against any
// other type of stub in the base model except for pointcut stubs. A stub
// in the pointcut expression that is not static must be matched against
// the same type of stub in the base model.
// An intentional element tagged with the anytype pointcut element may be
// matched to any type of intentional element in the base model. An element
// link tagged with the anytype pointcut element may be matched to any type
// of element link in the base model.
// Enhanced matching based on semantics: An AND-fork in the pointcut
// expression may be matched against a dynamic or synchronizing stub (when
// entering the stub). An OR-join in the pointcut expression may be matched
// against many end points with a plug-in binding for the same dynamic stub
// in the base model (when exiting the stub). An AND-join in the pointcut
// expression may be matched against many end points with a plug-in binding
// for the same synchronizing stub in the base model (when exiting the
// stub).
if !isTypeMatch(pcElement, baseElement) then
    return false;
endif
// The names of the model elements must match (this may require the use of

```

```

// conditions for the mapping, if the model elements are responsibilities
// or components). If the pointcut expression does not specify a name for a
// UCM model element except responsibilities, components, start points, and
// end points, the name is deemed to be the wildcard *.
if !isNameMatch(pcElement, baseElement) then
    if !isConditionalNameMatch(pcElement, baseElement) then
        return false;
    else
        defineCondition(pcElement, baseElement, condition);
    endif
endif
// The names of conditions must match, if the model elements are start
// points, waiting places, timers, failure points, or OR-forks. If the
// pointcut expression does not specify conditions, any condition may be
// matched.
if !isConditionMatch(pcElement, baseElement) then
    return false;
endif
// The metadata of the pointcut element must be a subset of the metadata of
// the base element.
if !isMetadataMatch(pcElement, baseElement) then
    return false;
endif
// A neighboring element must be reached from an element in the pointcut
// expression through the same type of node connection (regular, timeout,
// or trigger/release branch) and in the same direction as the neighboring
// base element from the base element.
if !isConnectionMatch(pcElement, baseElement) then
    return false;
endif
// match the containers of the pointcut element and the base element
if isGRLElement(pcElement) then
    // The GRL actor of the model elements must be compatible (i.e., for
    // elements within an actor to be matched, their actors also need to
    // match).
    URNmodelElement pcActor = pcElement.getActor();
    URNmodelElement baseActor = baseElement.getActor();
    if !isNameMatch(pcActor, baseActor) then
        return false;
    endif
    if !isMetadataMatch(pcActor, baseActor) then
        return false;
    endif
else
    // The component hierarchy of the model elements must be compatible
    // (i.e., for elements within a component to match, their components also
    // need to match; as components may be arranged in a hierarchy, the
    // immediate components may not need to be the same for a valid match, as
    // long as the component in the pointcut expression can be found in the
    // component hierarchy of the base element).
    URNmodelElement pcComponent = pcElement.getComponent();
    URNmodelElement baseComponent = findBaseComponent(pcComponent,
                                                    baseElement.getComponent());
    if !isNameMatch(pcComponent, baseComponent) then
        if !isConditionalNameMatch(pcComponent, baseComponent) then
            return false;
        else
            defineCondition(pcComponent, baseComponent, condition);
        endif
    if !isMetadataMatch(pcComponent, baseComponent) then
        return false;
    endif
    // A component of ComponentKind Team in the pointcut expression may be

```

```

// matched against any other type of component in the base model. A
// component in the pointcut expression that is not of ComponentKind Team
// must be matched against the same type of component in the base model.
if !isComponentTypeMatch(pcComponent, baseComponent) then
    return false;
endif
// The location of the model elements in their components must match
// (either first, last, or any location in the component).
if !isLocationInComponentMatch() then
    return false;
endif
endif // isGRLElement
return true;
end // matchElement

```

Operation: matchNeighbors
Input: URNmodelElement pcElement, URNmodelElement baseElement,
MatchList currentMatches
Output: MatchList
Exception: NoMatchFound (if match is unsuccessful)

```

begin // matchNeighbors
// A model element may not have any neighbors because it is not connected
// to any other element or may have already been visited.
if pcElement.getNeighbors() ==  $\emptyset$  then
    return currentMatches // Stops recursion.
endif
// Else, there are still path nodes to match.
MatchList result =  $\emptyset$ 
// In order to match the neighboring elements, all permutations of these
// elements have to be considered. Hence the need for the first for loop!
// The resulting permutations contain all possible mappings between all
// neighboring elements of pcElement and all neighboring elements of
// baseElement. These mappings can be accessed with getMappings() for each
// permutation. A match is successful if at least one permutation can be
// matched recursively.
// If the pcElement is the anything pointcut element and the neighboring
// elements cannot be matched, then a mapping from the anything pointcut
// element to the neighbor of the base element is returned, because the
// anything pointcut element matches any base element. In the next step of
// the recursion, the neighbour of that base element will be considered and
// the anything pointcut element is therefore incrementally matched against
// a sequence of base element until the neighbor of the anything pointcut
// element matches the neighbor of the base element. This approach also
// ensures that the shortest possible sequence of base elements with a
// given start element is matched against the anything pointcut element.
// The same technique as for the AoUCM anything pointcut element is used
// for GRL decomposition chains.
// Some neighbors may be skipped due to enhanced matching based on
// semantics (e.g., whitespace is skipped and stubs in the base model are
// matched only if there is a stub in the pointcut expression).
// Some neighbors may have alternatives due to enhanced matching based on
// semantics (e.g., an AND-fork may be matched against a dynamic stub).
// Alternatives are essentially handled by duplicating the portion of the
// model that pertains to the alternative, hence making both options
// available in the model to be matched.
MatchList pList = matchPermutations(pcElement, baseElement)
foreach Match p in pList
    MatchList permutationMatches =  $\emptyset$ 
    copy currentMatches to permutationMatches
    add p.getMappings() to permutationMatches

```

```

// After adding mappings from this permutation to the already established
// mappings for all established matches in currentMatches, check for each
// match whether the new mappings contradict the old mappings. If they
// contradict, remove the match from permutationMatches. In summary,
// contradictions are a) elements that cannot be matched against multiple
// other elements but are or b) illegal matches with ignored elements
// (e.g., a match where the beginning of a match is followed immediately
// by an ignored stub as described for the example in Figure 118).
remove all contradictory matches from permutationMatches
// If this permutation contradicts the established mappings, discard the
// permutation and move on to the next.
if permutationMatches == ∅ then
    continue with next loop iteration
endif
// If nothing has changed, then only duplicate mappings were added.
// Therefore, nothing is left to match and the recursion stops with the
// current matches as a result.
if permutationMatches is the same as currentMatches then
    result = currentMatches
else
    try
        MatchList mergeResult = ∅
        // For each mapping in the current permutation a recursive match has
        // to be attempted. Hence, the need for the second for loop! This
        // recursion effectively scans both the pointcut expression and the
        // base in parallel. There are two fail cases: 1) the recursive
        // match cannot find any matching elements (matchNeighbors() is not
        // successful) or 2) the merging of all mappings causes
        // contradictory mappings (merge is not successful).
        foreach Mapping pcElement2 to baseElement2 in p.getMappings()
            // Match recursively neighboring elements of pcElement2 against
            // neighboring elements of baseElement2 and return matches
            // (which contains permutationMatches plus new mappings found by
            // matchNeighbors()).
            // matchNeighbors() throws NoMatchFound exception if not
            // successful.
            MappingsList recursionResult = matchNeighbors(pcElement2,
                baseElement2, permutationMatches)
            // At this point, the results need to be merged. This merge is
            // necessary because in each pass of the for loop a branch is
            // explored recursively and a match is found only if the results
            // from all branches together make sense (i.e., they do not
            // contradict each other). Merge also throws NoMatchFound
            // exception if not successful.
            merge recursionResult with mergeResult
        endforeach // Mapping
        add mergeResult to result
    endtry
    catch NoMatchFound
        // Do nothing and continue for loop (this gives the next permutation
        // a chance).
    endcatch
endif // Same as currentMatches
endforeach // Match
if result == ∅ then
    // Before throwing an exception, all matches with the anything pointcut
    // element need to be saved to a global variable for post-processing.
    // This is required to address the situation described in Figure 115.
    save all matches in currentMatches that contain anything pointcut element
    throw new NoMatchFound()
endif
return result
end // matchNeighbors

```

Appendix E: Composition Algorithm

A high level summary of the composition algorithms for AoGRL and AoUCM follows, demonstrating the composition of a single aspect with the base model. Both algorithms take as input the list of matches containing mappings for a pointcut diagram. This input is established by the matching algorithm for the aspect. The result of the algorithms is an updated URN model to which aspect markers and other metadata have been added. In addition to the operations `composeGRLAspect` and `composeUCMAAspect`, several other operations are required. Descriptions of these operations can be found in the comments below.

```
Operation:   composeGRLAspect
Input:      PointcutGraph pcGraph, URNmodel base, MatchList matchList
Output:    URNmodel
Exception: CompositionNotRequired

begin // composeGRLAspect
URNmodel result = base
Boolean updateFlag = false
// Check whether the special case occurs, i.e., pointcut expression of the
// pointcut graph is not connected to any non-pointcut elements, the
// pointcut deletion marker is not used on the pointcut graph, and a
// disjoint non-pointcut element exists on the pointcut graph. If yes,
// return true. Otherwise, return false.
Boolean specialCase = checkForSpecialCase(pcGraph)
foreach Match match of matchList
    Boolean aspectMarkerAdded = false
    // Step 1 of composition for AoGRL models as well as the first part of
    // Step 2 (mapping to pointcut graph)
    foreach Mapping m of match
        // Add an aspect marker to the definition of the base element in the
        // result, if the special case occurs or if the pointcut element in the
        // mapping is connected to a non-pointcut element or a pointcut element
        // tagged with the pointcut deletion element (i.e., the aspect
        // transforms the base element).
        // Furthermore, also convert the mapping to metadata if an aspect
        // marker is added and add the metadata to the definition of the
        // base element in the result.
        if specialCase || isTransforming(m.getPointcutElement()) then
            add metadata for aspect marker to m.getBaseElement()
            add metadata for mapping to pointcut graph to m.getBaseElement()
            aspectMarkerAdded = true
        endif
    endforeach // Mapping
    // Second part of step 2 of composition for AoGRL models (substitution
    // instructions)
    if aspectMarkerAdded then
```

```

    foreach Mapping m of match
        // If the pointcut element is parameterized, add metadata for the
        // substitution instructions to the pointcut graph in the result.
        if isParameterized(m.getPointcutElement()) then
            add aspect marker for substitution to pcGraph
        endif
    endforeach // Mapping
    updateFlag = true
    endif // aspectMarkerAdded
endforeach // Match
if !updateFlag then
    throw new CompositionNotRequired()
endif
return result
end // composeGRLAspect

```

The operation `composeUCMAAspect` makes use of several data structures to keep track of the mappings created for aspect markers and variables. The data structure `AspectMarkerMappings` contains a mapping for the in-binding, a mapping for the out-binding, and the node connection that identifies the insertion point. Furthermore, it contains a flag indicating whether all required mappings have been created. The data structure `VariableMapping` contains a mapping for a variable. `composeUCMAAspect` reflects the various steps in the AoUCM composition algorithm, and is thus composed of several other operations, i.e., `composeUCMAAspectPreprocessing`, `composeUCMAAspectStep1`, `composeUCMAAspectStep2`, and `composeUCMAAspectStep3`.

```

Operation:   composeUCMAAspect
Input:      AspectMap aMap, PointcutMap pcMap, URNmodel base,
               MatchList matchList
Output:    URNmodel
Exception: MalformedAspectMap, CompositionNotRequired

begin // composeUCMAAspect
URNmodel result = base
Boolean updateFlag = false
// Determine whether interleaved composition occurs by checking whether
// more than one pointcut stub exist on a path of the aspect map.
Boolean interleaving = isInterleaving(aMap)
// Determine whether a replacement composition occurs by checking whether
// the replacement pointcut stub exists on the aspect map.
Boolean replacement = isReplacement(aMap)
foreach Match match of matchList
    // Determine whether the match crosses map boundaries, i.e., at least one
    // start or end point of the pointcut expression resides on a different
    // maps. Note that is irrelevant for composition if anything between the
    // start and end points resides on different maps - only the start and
    // end points are important.
    Boolean crossing = isCrossingMapBoundaries(match)
    AspectMarkerMappingsList amm = ∅
    VariableMappingList vm = ∅
    // Preprocessing
    AspectMarkerList amList = ∅
    StubList stubList = ∅

```

```

String condition = ""
composeUCMAAspectPreprocessing(replacement, crossing, match, amList,
                                stubList, condition)
// Step 1
composeUCMAAspectStep1(interleaving, crossing, amm, match, vm, aMap)
// Step 2
Boolean amGroupRequired = false
// may throw exception
composeUCMAAspectStep2(interleaving, amm, aMap, pcMap, amGroupRequired)
// Step 3
composeUCMAAspectStep3(interleaving, replacement, crossing,
                        amGroupRequired, amm, vm, aMap, amList,
                        stubList, condition, result)
// Postprocessing: optimization of tunnel entrance/exit aspect markers
// that follow each other immediately into regular or conditional aspect
// marker in result
optimize(result)
updateFlag = true
endforeach // Match
if !updateFlag then
    throw new CompositionNotRequired()
endif
return result
end // composeUCMAAspect

```

Operation: `composeUCMAAspectPreprocessing`
Input: Boolean interleaving, Boolean crossing, Match match,
AspectMarkerList amList, StubList stubList, String condition
Output: ---
Exception: ---

```

begin // composeUCMAAspectPreprocessing
// Preprocessing: copy mappings to unnamed pairs of connected end/start
// points in the pointcut expression if interleaved composition occurs
if interleaving then
    add mappings for connected end/start points to match
endif
// Preprocessing: scan match for existence of aspect markers with
// provided group tags
amList = findAspectMarkersWithProvidedGroup(match)
// Preprocessing: retain map hierarchy but only if the match is crossing
// map boundaries
if crossing then
    stubList = findStubs(match)
endif
// Preprocessing: collect all conditions from the mappings
condition = collectConditions(match)
end // composeUCMAAspectPreprocessing

```

Operation: `composeUCMAAspectStep1`
Input: Boolean replacement, Boolean crossing, AspectMarkerMappingsList
amm, Match match, VariableMappingList vm, AspectMap aMap
Output: ---
Exception: ---

```

begin // composeUCMAAspectStep1
foreach in-path and out-path inout of pointcut stub of aMap
// Step 1.a: ignore inout if its path segment is empty and replacement
// does not occur and crossing map boundaries does not occur
if residesOnEmptySegment(inout) && !replacement && !crossing then
    continue with next loop iteration

```

```

endif
add empty AspectMarkerMappings for inout to amm
// Step 1.b: create the mapping of inout with the base element based on
// the existing mappings and the plug-in bindings of the pointcut stub.
// Special case for named start/end points is considered.
// createFirstMapping returns true if successful and false if not.
if !createFirstMapping(amm[inout], match) then
    remove AspectMarkerMappings[inout] from amm
    continue with next loop iteration
endif
// Step 1.c: establish the insertion point for the base element (for
// named start/end points the insertion point is null, indicating the
// location before/after the element)
findInsertionPoint(amm[inout], match)
// Step 1.d: establish the mappings for variables
createVariableMapping(vm, aMap, match)
endforeach // Step 1
end // composeUCMAAspectStep1

```

Operation: composeUCMAAspectStep2
Input: Boolean interleaving, AspectMarkerMappingsList amm,
AspectMap aMap, PointcutMap pcMap, Boolean amGroupRequired
Output: ---
Exception: MalformedAspectMap

```

begin // composeUCMAAspectStep2
// Step 2: scanning
// Scan the aspect map from an in-path or out-path of the pointcut stub
// to discover the start or end point, respectively. For out-paths, the
// aspect map is scanned forwards. For in-paths, the aspect map is scanned
// backwards. scan returns start points, end points, and pointcut stubs
// but does not return local start/end points.
foreach in-path and out-path inout of pointcut stub of aMap
    URNmodelElementListList el[inout] = scan(inout)
endforeach // Step 2: scanning
// Step 2: creating mappings
repeat
    Boolean nothingChanged = true
    Boolean allDone = true
    foreach in-path and out-path inout of pointcut stub of aMap
        // Only try to create mapping, if we have not yet found the mapping.
        if !amm[inout].isAllCreated() then
            // Attempt to create the second mapping based on the elements
            // returned by the scan. createSecondMapping returns true if
            // successful and false if not.
            if createSecondMapping(amm[inout], el[inout]) then
                nothingChanged = false
                continue with next loop iteration
            endif
            // If we get to this point, we were not successful in creating the
            // required second mapping and we are not yet done with this
            // inout.
            allDone = false
        endif // allCreated
    endforeach // inout
    // If unsuccessful, remove those start and end points claimed by other
    // mappings in this repeat loop. This will allow other mappings to be
    // created in the next repeat loop.
    if !allDone && !nothingChanged then
        remove claimed start/end points from el
    endif
until nothingChanged || allDone // Step 2: creating mappings

```

```

// If there are still in/out-paths for which all required mappings could
// not be established, the aspect map is malformed.
if !allDone then
    throw new MalformedAspectMap()
endif
// The partial order check for interleaving examines start and end points
// with multiple plug-in bindings. For start points, only the binding
// with the first pointcut stub is allowed. For end points, only the
// binding with the last pointcut stub is allowed.
if interleaving then
    ensurePartialOrdering(amm, aMap, pcMap)
endif
// Scan aspect map from each entry point. If a scan of entry point E with
// aspect marker A leads to an element that does not have an out-binding
// to A but does have an out-binding to a different aspect marker, an
// aspect marker group is required. Replacement pointcut stub and
// interleaving automatically require aspect marker groups.
amGroupRequired = determineAMGroupNeeded(amm, aMap)
// If an end point has out-bindings to multiple aspect markers and an
// aspect marker group is required, the aspect map is malformed.
if existsEndPointWithoutBindings(amm, aMap) && amGroupRequired then
    throw new MalformedAspectMap()
endif
// Issue warning if start or end points of the aspect map are not local
// and do not have a plug-in binding with an aspect marker.
if !allStartEndpointsUsed(amm, aMap) then
    issue local start/end points warning
endif
end // composeUCMAAspectStep2

```

Operation: composeUCMAAspectStep3
Input: Boolean interleaving, Boolean replacement, Boolean crossing,
Boolean amGroupRequired, AspectMarkerMappingsList amm,
VariableMappingList vm, AspectMap aMap, AspectMarkerList amList,
StubList stubList, String condition, URNmodel result

Output: ---
Exception: ---

```

begin // composeUCMAAspectStep3
    foreach in-path and out-path inout of pointcut stub of aMap
        // Step 3.a: add an aspect marker for inout and convert the mappings
        // for inout into plug-in bindings in the result. A scan first
        // determines if a conditional aspect marker is required. Dynamic
        // aspect markers are also considered.
        Boolean cond = existsConditional(aMap, amm[inout].getEntryPoint())
        addAspectMarker(result, amm[inout], replacement, interleaving, cond)
        // Step 3.b: if an aspect marker group is required, add metadata to the
        // aspect marker of this in/out-path in the result
        if amGroupRequired then
            add metadata for aspect marker group to aspect marker
        endif
        // If the match is crossing map boundaries, then provided group is
        // added for aspect marker with an in-binding to a start point, while
        // required group is added to all other aspect markers.
        // Furthermore, the map hierarchy of the match is retained by adding
        // the list of stubs to any aspect marker that has an out-binding to an
        // exit point. This, however, is necessary only if a replacement or
        // conditional composition occurs.
        if crossing then
            add metadata for provided/required group to aspect marker
            if replacement || cond then
                add stubList to aspect marker // only for exit points!
        endif
    endfor
end

```

```

    endif
  endif
  // If aspect markers with the provided group tag exist in the match and
  // the match is either a replacement or a conditional, then transfer
  // the tag to this aspect marker but only if the aspect marker has an
  // in-binding to an entry point.
  if !amList.isEmpty() && (replacement || cond) then
    add amList to aspect marker // only for entry points!
  endif
  // If condition because of variables, component plug-in binding, or
  // responsibility plug-in binding exists, add it to the aspect marker
  if !condition.isEmpty() then
    add condition to aspect marker
  endif
endforeach // Step 3
// Step 3.c: convert the mappings for variables into component and
// responsibility plug-in bindings in the result
convertVariableMappings(result, vm)
end // composeUCMASpectStep3

```

Operation: createSecondMapping
Input: AspectMarkerMappings ammForInout, URNmodelElementList elForInout
Output: Boolean
Exception: ---

```

begin // createSecondMapping
  // If one start or end point is found regardless of the number of pointcut
  // stubs, create second mapping.
  if elForInout.containsOnlyOneStartOrEndPoint() then
    ammForInout.setSecondMapping(elForInout.getStartOrEndPoint())
    ammForInout.setAllCreated(true)
    return true
  endif
  // If no start or end point is found but pointcut stubs, do not create a
  // second mapping but consider the inout done.
  if elForInout.containsOnlyPCStubs() then
    ammForInout.setAllCreated(true)
    return true
  endif
  // Otherwise, a) more than one start or end point was found or b) nothing
  // was found at all.
  return false
end // createSecondMapping

```