UNIVERSITY OF LUXEMBOURG
FACULTE OF SCIENCES, TECHNOLOGY & COMMUNICATION

&

UNIVERSITY OF OSNABRUECK
FACULTY OF INFORMATICS

DOCTORAL SCHOOLS

P H D T H E S I S

to obtain the title of

Docteur en Informatique

of the University of Luxembourg

&

Dr. rer. nat.

of the University of Osnabrueck

Defended by

Marwane EL KHARBILI

Enterprise Management of Regulatory Compliance

A Model-Driven Framework for Policy-Based Regulatory Compliance Management in Business Process-Centered Enterprise Models

defended on July 22, 2013

Thesis Committee :

Chairman:	Prof. Dr. Leendert van der Torre	-	University of Luxembourg
Vice-Chairman:	Dr. Guido Governatori	-	NICTA Queensland, Australia
Supervisors:	Prof. Dr.-Ing. Elke Pulvermueller	-	University of Osnabrück
	Prof. Dr. Pierre Kelsen	-	University of Luxembourg
Members:	Prof. Dr. Henderik Proper	-	Radboud University Nijmegen
	Junior-Prof. Dr.-Ing. Martin Kada	-	University of Osnabrück

"Be sure We shall test you with something of fear and hunger, some loss in goods, lives, and the fruits of your toil. But give glad tidings to those who patiently persevere. Those who say, when afflicted with calamity, 'To Allah we belong, and to Him is our return.' They are those on whom descend blessings from their Lord, and mercy. They are the ones who received guidance."

Holy Qur'ân (Sura: 2 - Âyates: 155-157)

To Fatima, my dear aunt, who left us in December 2011. May God let her rest in a place of divine light and fresh breeze.

To my family and especially my mother, Zohra, the light of my life, with her infinitely loving heart. Her sacrifices made me into the man I am today, and the man I will grow further to become. She tirelessly fought through the toughest economic and family conditions, to allow me to achieve educational excellence. She is my biggest supporter, always believed in me and has never let me down. She will always be there with me every step of the way.

To Issame, Manale and Halima, my siblings who fill my life with the warm feeling of being loved and accepted for who I am. Without them my childhood and youth would have been a long and solitary desert crossing.

To Rajâa. She bared with me through the tough times where I was continuously on the edge, absent, irritable, sometimes even despicable. She showed me the true meaning of affection, friendship and dedication.

Declaration of Integrity

This work was done under the guidance of Professors Pierre Kelsen and Elke Pulvermüller, under a co-tutelle agreement between the universities of Luxembourg and Osnabrück.

This dissertation is a presentation of my original research work and was carried out in accordance with the Regulations of the Universities of Luxembourg and Osnabrück. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions. Any views expressed in the dissertation are those of the author and in no way represent those of the Universities of Luxembourg and Osnabrück.

Marwane El Kharbili

Signature: .....

Date:

In our capacity as supervisors of the candidate's thesis, we certify that the above statements are true to the best of our knowledge.

Prof. Dr. Pierre Kelsen

Signature:.....

Date:

Prof. Dr.-Ing. Elke Pulvermüller

Signature:.....

Date:

Foreword

The dissertation is just the infant of a woman called research. It is not its masterwork, but it is as special as the first child. During research, you somehow know there is a road ahead, but a road that is so dark that it is very hard to see ahead with anything else than intuition. In many ways, research is like creating songs and music. Only the artist's intuition makes him an appreciated artist. Doing research is like writing the lyrics, and writing the dissertation is like singing the lyrics to a music that seduces the reader. In this regard, I compare a dissertation to an artist's very first album.

It requires mastering all the musical instruments of course, the scientist's tools, the communication, presentation and writing techniques, the scientific method. Still, making good music most of all requires listening to all kinds of music, songs and inspirations. Let all that boil into the artist's head, drop a good pinch of innovation, creativity and instinct into it, use a great deal of faith and conviction to do mix, and that will somehow make a dissertation. One cannot write a dissertation from scratch, and it is most often not an radical innovation, it is part of a continuum of human creativity.

> *All you do is sit down at a typewriter and bleed.*
> - Ernest Hemingway.

> *Writing is easy, all you do is sit staring at a sheet of paper until the drops of blood form on your forehead.*
> - Gene Fowler.

A Dissertation writing process is chaotic and unordered. I believe that the old adage "*failing to plan is planning to fail*" does not apply to thesis dissertation writing. Every principle, even the most broadly accepted, has its exceptions. The process of dissertation writing is more like daring to stare at the remaining top 15% of a mountain that it took years to prepare to climb. The difference is that the high altitude leaves you with almost no oxygen and you can barely stay on your feet. A blizzard just decides to make things harder and temperature freezes while you get hit by big chunks of snow.

But you can't stop, not now, not after all that you have been through. All the hurdles, detours, challenges along the way, all the comrades that you lost, all those people that look up to you, there has got to be a reason for all of this. Deep inside, you know, it is now or never, your body and mind won't take it for long anymore, there is just nothing left, you are emptied of your last drops of illusion and all that is left is the energy of despair. If you back down, it is over. If you look ahead it is over. But at that one point where you reach the climax of despair, this sudden and strong thought comes right at your face: "*WALK!*".

So somehow, although you heard it zillions of times from other people and did nothing about it, you keep your head down and walk in humility, because you know it, this time it's different. It is that feeling of utter respect for the haunting task ahead that you feel it deep down inside of you, this true and absolute modesty that now words can in any common measure describe accurately.

That is how it feels. Now it is over, and you are at the top of your mountain.

Acknowledgments

The adventure of this thesis was only a success because of the understanding, support and openness and much patience of many people through twenty-five years. To all those who were involved with me along this path, whether you are named in these lines underneath or not, you have my eternal recognition.

I first of all thank Elke Pulvermüller for having been a scientific companion since the beginning and a true confident and supporter. I am equally deeply obliged to Pierre Kelsen for having offered me the opportunity to work in his lab, and directly supervise my scientific training and dissertation. His patience and critical stance pushed me to dive into the world of formal methods. Much of what I learned as a researcher I learned through him. I had access to every resource a researcher needs to excel in his work. Elke and Pierre trusted my ambitious aims, brought me gently to the ground of research, and always knew how to handle my love for multi-tasking and my "idea storms". Elke and Pierre have been a lot more than supervisors, they know many of my secrets, acted like trusted friends, supportive through the very tough phases of the thesis when I was on the edge. They are the reason I stayed focused on THE objective.

Many other people were involved along the adventure, continuously feeding me with new knowledge, ideas, honest and non-complacent feedback and very valuable critics. Moussa Amrani was involved in this thesis in a special way, and must be treated in a special way. Moussa was my guide in the world of formal methods, language semantics and formal verification. Moussa possesses very wide and deep knowledge of these topics, and was very often there to help me grasp every subtle detail of many theoretic tools. Although he also has a thesis to work on, he took time to get me started in many topics around formal modeling, MDE, and verification. Moussa has also been my office mate for almost four years, and was my teammate in office group prayers and my ride to the mosque on Fridays for more times than I could count.

Qin ma, my co-investigator in the MaRCo project, always gave me very critical, thorough and precise feedback that substantially enhanced the quality of my work. Thanks to her expertise in language engineering and formal modeling, my discussions with her certainly flew into the thesis. Guido Governatori was also an instrumental person in my thesis, in his quality of an internationally recognized expert in compliance. I thank him very much for his advice on scoping the contribution of this thesis and very enriching collaboration on modeling and checking obligations. Christian Glodt is the person everyone reading about or using CoReL should thank. His many years of experience with the Java technologies and the Eclipse framework, the EMF/GMF framework and model transformation languages have enabled tool support for this thesis to be of the quality it is. Without him, the prototypes would be functional, but very limited and rudimentary.

Numerous people have been involved indirectly with this thesis, some of them are professors, some Post-docs, and some fellow PhD students. Many ideas and discussions triggered further developments of the work contained in this thesis. I will cite some of them here, and ask the others I may not have mentioned to forgive me: Sebastian Stein, Andreas Rusnjak, Wolfgang Runte, Ivan Markovic, Agata Filipowska, Monika Kaczmarek, Piotr Stolarski, Veronica Gacitua-Decar, Ken Decreus, Levi Lucio, Vasco de Sousa, Leon van der Torre, Guido Boella, Andreas Speck, Vasco Amaral, Silvano ColomboTosatto, Alfredo Capozucca. I also thank our

secretary, Daniele Flammang for the many times she saved my life and having a great sense of humor, but especially for understanding mine. My special recognition goes to Dr. Nenad Stojanovic for introducing me to world-class research, in the field of ontologies and the semantic web, at the FZI research center in Karlsruhe where he supervised my master thesis.

It has been a privilege to have been surrounded and have met so many incredibly brilliant people. I dearly wish my path in research will lead me to continue collaborating with these people and hope to continue to meet equally intriguing and intellectually exciting people. Like all the best things, my adventure in research an coincidence that almost never happened but an unexpectedly enriching experience. Through all the ups and downs I learned as much about myself as about research.

I also want to thank the Moroccan government for having funded my studies in France at the Grenoble Institute of Technology under an academic excellence scholarship. My gratitude to my country for granting me this honor is infinite. Also, I would like to thank both the Rhône-Alpes region and the French-German University for having partly financed my double M.Sc. degree at the Karlsruhe Institute of Technology. Last but not least, all this would certainly not have been possible without the trust and support of Jean-louis Roch and Marianne Genton at the ENSIMAG school of the Grenoble Institute of Technology.

My deepest recognition goes to the teachers I had in primary, secondary and high schools. I forgot names but I can mention Mrs. Ben El Malek, Mrs. Kawkab, Mrs. Addou, Mr. El Houzali and especially Mrs. Laklalech. I can never forget the passion and efforts they put in keeping a highly energetic and a dreaming young boy on track. They have seen potential in me, and I have since been trying to confirm their hopes.

I am deeply thankful, on a personal level, to 'aunt' Elkabira, 'uncle' Ahmad, their two sons Hicham and Mohammed El Mouatassim, to 'Khalti Essâdiya', for having been there for the family in the tough and the happy times alike. They supported my mother who suffered because of her health when her children were not there for her. They are family to me.

Much of this work was conducted under the MaRCo project on regulatory compliance, under a luxembourgish FNR funding scheme. Also, many of the core ideas this thesis is based on, originated during my involvement in the IP-SUPER project on semantic business process management while I was a research engineer at the IDS Scheer AG (now Software AG).

Abstract

Regulatory Compliance Management (RCM) is widely recognized as one of the main challenges still to be efficiently dealt with in Enterprise Models (EMs). In the discipline of Business Process Management (BPM) in particular, which plays a central role in modern management of enterprises, compliance is considered as an important driver of the efficiency, reliability and market value of companies. It consists of ensuring that enterprise systems behave according to some guidance provided in the form of regulations.

Existing approaches to RCM tackle this issue from two different perspectives: methodological and formal. The first category of a pproaches is widely used in the industry and the practitioner community and proposes several processes based on controls for compliance audit and governance. The second category of approaches seeks to construct complex formal languages and reasoning engines for automatically deciding on the state of compliance of a business process, but remains hardly accessible to practitioners who are not trained in formal methods. The goal of this thesis is to provide an approach for modeling and checking of regulatory compliance that profits from the power of complex formal languages and is specifically targeted at practitioners.

In this thesis, we first give a working definition of the RCM research problem, and define the scope covered in our research. Then, we analyze the RCM challenge for business process-centered enterprise models by describing the challenges that compose RCM. Based on the latter we propose a conceptual framework for RCM.

In order to answer the research challenges identified, we show why we expect a formal policy-based and model-driven approach to provide significant advantages in allowing enterprises to flexibly manage decision-making related to regulatory compliance.

For this purpose, we contribute CoReL, a visual domain-specific modeling language for representing compliance requirements. The main objective of CoReL is to bring the task of compliance modeling to the business user level where it belongs. CoReL allows to leverage business process compliance modeling and checking, enhancing it with regard to, user-friendliness and coverage of various enterprise artifacts, as well as various types of regulatory constraints. Both informal and formal semantics of CoReL are introduced and its use for modeling and checking compliance regulations is shown on a set of examples using a specifically developed tool.

Keywords Regulatory Compliance, Business Processes, Policies, Model-Driven Engineering, Verification, Enforcement, Alloy.

Contents

I Domain Analysis **1**

1 Introduction **3**
1.1 Context . . . 3
1.2 Compliance in the Context of Business Process Management . . . 5
1.2.1 Shortcomings of Regulatory Compliance Management Methods . . . 6
1.3 Thesis Design . . . 9
1.3.1 Thesis Scope . . . 9
1.3.2 Thesis Objectives . . . 10
1.3.3 Research Questions . . . 10
1.3.4 Results . . . 11
1.3.5 What This Thesis Is Not . . . 12
1.3.6 What This Thesis Is . . . 12
1.4 Structure Of The Thesis . . . 12
1.5 Summary . . . 13

2 Background **15**
2.1 Business Process Management . . . 15
2.1.1 Business Processes . . . 15
2.1.2 Business Process Lifecycle . . . 16
2.1.3 (Enterprise) Business Process Modeling . . . 18
2.1.4 Business Process Modeling Languages . . . 19
2.2 Model-Driven Engineering . . . 20
2.2.1 The Need for Abstraction . . . 20
2.2.2 The (Meta-) Modeling Approach . . . 21
2.2.3 Basic Ingredients for Metamodel-Based Language Engineering . . . 22
2.2.4 The Engineering of Metamodeling . . . 23
2.2.5 Formal Verification . . . 25
2.3 Alloy . . . 26
2.3.1 Introduction . . . 26
2.3.2 Syntax and Semantics . . . 26
2.3.3 The Alloy Analyzer . . . 30
2.3.4 Comparison of Alloy with Other Formal Methods . . . 31

3 State of the Art in Regulatory Compliance Management **33**
3.1 Introduction . . . 33
3.2 Background Definitions . . . 34
3.3 Conceptual Framework for Business Process RCM . . . 36
3.3.1 The Business Process Management Dimension . . . 36
3.3.2 The Compliance Dimension . . . 37
3.4 Analysis Design . . . 41
3.5 Overview of Selected Publications . . . 42
3.6 A Comparative Analysis of Solutions to RCM for BPM . . . 46
3.6.1 End-User, Application and Violation Management . . . 47
3.6.2 Rule Language Expressiveness, Business Aspects and Policy Powers . . . 49

3.7 Discussion . . . 50
3.8 Summary . . . 52

II CoReL Framework for Regulatory Compliance 53

4 The System Model 55
4.1 The ASE Model . . . 55
4.2 The SBP Model . . . 57
4.2.1 Background Definitions . . . 57
4.2.2 Example: Printer . . . 60
4.3 Formalizing the System Abstract Syntax Metamodel . . . 61
4.3.1 Examples of Defining SBP Models . . . 69
4.3.2 Example of Defining a System Model . . . 74
4.4 Formalizing the System Execution Semantics . . . 76
4.4.1 Example of Executing a Simple System Model . . . 80

5 A Policy-Based Approach to Regulatory Compliance Management 85
5.1 CoReL Rationale . . . 85
5.1.1 Another Look at the RCM Lifecycle . . . 85
5.1.2 Business Policies as a Semantic Bridge . . . 86
5.1.3 Language Engineering . . . 87
5.2 A Deeper Look at Compliance Requirements . . . 89
5.2.1 The Semantic Gap in Existing RCM Solutions . . . 89
5.2.2 Coping with Regulation Complexity . . . 90
5.2.3 2D Classification of CRs . . . 90
5.2.4 Coverage of 2D CR Space . . . 92
5.3 A Running Example: Printer Usage . . . 92
5.3.1 The Example Process . . . 92
5.3.2 Compliance Rules . . . 93
5.4 Abstract Syntax . . . 94
5.4.1 Abstract Syntax . . . 94
5.5 Informal Semantics of CoReL Policy Interpretation . . . 103
5.5.1 Policy Governance . . . 103
5.5.2 Enforcement of Authorization Policies . . . 105
5.6 Modeling A Simple Business Process Printer Management Regulation . . . 106

6 Semantic Foundations of CoReL 113
6.1 Formalizing CoReL . . . 113
6.1.1 Formalizing the CoReL Abstract Syntax Metamodel . . . 113
6.1.2 Formalizing the CoReL Execution Semantics . . . 115
6.2 Applying of the Formalization: Printer Example - Access Control . . . 120
6.2.1 The Business Domain Model . . . 121
6.2.2 The System Model . . . 122
6.2.3 The Compliance Model . . . 122

7 CoReL Concrete Syntax 125
7.1 The MaRCo Tool Suite . . . 125
7.2 CoReL Symbols . . . 126
7.2.1 Symbols Table . . . 126

7.3 CoReL Diagrams . . . 129
7.3.1 Regulation Diagram . . . 129
7.3.2 Policy Diagram . . . 131
7.3.3 Violation Diagram . . . 131
7.3.4 Remarks on Diagrams . . . 132

III Regulatory Compliance Verification 133

8 CoReL's Approach to Verification 135
8.1 Creating System Models . . . 135
8.1.1 A DSL for System Models . . . 135
8.2 Description of the Approach . . . 137
8.2.1 Binding CoReL and System Models . . . 137
8.2.2 Alloy-based Verification . . . 139
8.3 Model Transformations . . . 139
8.3.1 SBP to Alloy . . . 139
8.3.2 Regulation to Alloy . . . 144
8.3.3 CoReL to Alloy . . . 147
8.3.4 Violation Decision to Alloy . . . 149

9 Validation: Case Studies 155
9.1 Compliance Requirements Dimensions . . . 155
9.2 The Printer Access Control Case . . . 157
9.2.1 The Compliance Case . . . 157
9.2.2 The Compliance Requirements . . . 159
9.2.3 CoReL Engineering of the Compliance Requirements . . . 159
9.3 The Expense Refund Case . . . 172
9.3.1 The Process Case . . . 172
9.3.2 The Compliance Requirements . . . 172
9.3.3 Engineering the CoReL Models . . . 173
9.4 Modeling an Article of the HIPAA Regulation . . . 180

IV Finale 183

10 Conclusion 185
10.1 Research Summary . . . 185
10.1.1 Synopsis . . . 185
10.1.2 Results . . . 186
10.2 Contributions . . . 187
10.3 Discussion . . . 187
10.3.1 Challenges Faced & Lessons Learned . . . 187
10.3.2 Limitations of the Work . . . 189
10.3.3 Comparison with other Approaches . . . 191
10.4 Perspectives for Future Research . . . 193
10.5 Conclusion . . . 194

A Publications 197
A.1 Appendix I-Publications Produced by the Author . . . 197

Bibliography 201

List of Figures

1.1 The lifecycle of regulatory compliance management . . . 4
1.2 Thesis Structure . . . 13

2.1 BPM Lifecycle - Adapted from [zM04] - Extended with RCM Steps Contributed in this Thesis . . . 17
2.2 The OMG MDA Pyramid (from [tow07]) . . . 21
2.3 Model Transformations in Language Engineering (from [Kle08]) . . . 23
2.4 Alloy Analyzer Instance for The LightState Model . . . 30
2.5 Alloy Analyzer Counter Example to the lightsAreAlwaysAllRed Assertion . . . 31

3.1 Automated Regulatory Compliance Management - Conceptual Framework . . . 36

4.1 ASE Triple . . . 55
4.2 System Execution Model . . . 56
4.3 Do | Doing | Done Predicates in System Execution . . . 57
4.4 Process Example . . . 59
4.5 Simple Printer Example . . . 60
4.6 Simple Printer Example - Block Breakdown . . . 61
4.7 The ASE Metamodel . . . 62
4.8 The SBP Metamodel . . . 63
4.9 Sequence Block . . . 70
4.10 Result of Running an Alloy Model - sbp1 Example . . . 71
4.11 Sequence with Xor Block . . . 71
4.12 Xor Block of Sequences . . . 72
4.13 Nested Sequence and And Blocks . . . 73
4.14 Result of Running an Alloy System Model - sbp1 Example . . . 75
4.15 Result of Running an Alloy System Model with defined ASEs - sbp1 Example . . . 76
4.16 ExecState . . . 77
4.17 Process Execution Trace - Tree View . . . 81
4.18 Process Execution Trace - Projection on ProcessState - First State . . . 81
4.19 Process Execution Trace - Projection on ProcessState - Second State . . . 82
4.20 Process Execution Trace - Projection on ProcessState - Third State . . . 82
4.21 Process Execution Trace - Projection on ProcessState - Fourth State . . . 83

5.1 Simplified Regulatory Compliance Management Lifecycle- Interaction with a BPM Lifecycle . . . 86
5.2 3 Worlds Divide - Regulation, Enterprise and Decision Domains . . . 87
5.3 Semantic Gap Between Compliance and Logic Modeling . . . 89
5.4 The printer Example . . . 93
5.5 CoReL Metamodel [in ECORE] . . . 95
5.6 Example of a Business Model Extension of ASE . . . 97
5.7 Violation Type - Lights - Discrete Set of Violation Values . . . 99
5.8 Violation Diagram . . . 100
5.9 Process Compensation Illustration . . . 102
5.10 ASE Triple - Predicates . . . 103
5.11 Policy State Machine - Context Semantics . . . 104

5.12 Policy State Machine - Decision Making & Propagation . . . 104
5.13 CoReL Policy - Permission State Machine - Single Violation . . . 105
5.14 CoReL Policy - Permission State Machine - Multiple Violations . . . 106
5.15 CoReL - Printer Example - Regulation Model . . . 107
5.16 CoReL - Printer Example - Accessing an Assigned Policy Model . . . 107
5.17 CoReL - Printer Example - Modeling CR1 . . . 108
5.18 CoReL - Printer Example - Modeling CR1 - second alternative . . . 108
5.19 CoReL - Printer Example - Modeling CR1 - External Students . . . 108
5.20 CoReL - Printer Example - CR2 . . . 109
5.21 CoReL - Printer Example - Corrected CR2 . . . 109
5.22 CoReL - Printer Example - CR3 . . . 110
5.23 CoReL - Printer Example - CR4 . . . 110

6.1 Compliance Trace - Illustration . . . 119

7.1 The Editor Architecture of the MaRCo Tool . . . 125
7.2 Sarbanes Oxley Act 2002 - Regulation Model Excerpt . . . 130
7.3 CoReL - Multiple Regulations - Compliance Requirement Reuse . . . 130
7.4 CoReL - Printer Example - Random Permission Policy . . . 131
7.5 Violation Diagram . . . 132

8.1 xBPMN metamodel in ECORE . . . 136
8.2 The printer Example . . . 136
8.3 Semantic Domain Bridging Through CoReL . . . 137
8.4 ASE Binding at metamodel Level - Example . . . 138
8.5 CoReL's Approach to Design-Time Verification Using Alloy . . . 140
8.6 Multiple Pools Rule . . . 141
8.7 SBP to Alloy Transformation Pattern Rules . . . 142
8.8 Mapping Blocks - Examples . . . 143
8.9 Mapping Sending/Receiving Messages . . . 144
8.10 Regulation to Alloy Transformation Pattern Rules . . . 146
8.11 CoReL Policy Model to Alloy Transformation Rules . . . 148
8.12 Violation Model to Alloy Transformation Rules . . . 150
8.13 Excerpt of a Violation Model . . . 151
8.14 Excerpt of a Violation Model . . . 153

9.1 The Three Dimensions of Compliance Requirements Modeling in CoReL . . . 156
9.2 The printer Example . . . 157
9.3 The printer Example - Expanded Complex Tasks . . . 158
9.4 The printer Example - Conceptual Illustration of the Corresponding Alloy System Model . . . 158
9.5 CR1 - Policy Model . . . 160
9.6 CR1 - Alternative Policy Model - Prohibition . . . 161
9.7 CR2 - Policy Model . . . 163
9.8 CR2 - Violation Model . . . 164
9.9 CR3 - Policy Model . . . 166
9.10 CR3 - Violation Model . . . 166
9.11 Violation Model for CR1, CR2 & CR3 . . . 167
9.12 CR1/CR2/CR3 - Alternative Policy Model - Prohibition . . . 167
9.13 CR1/CR2/CR3 - Alternative Violation Model - Prohibition . . . 168

9.14 CR5 - Policy Model . . . 169
9.15 CR5 - Violation Model . . . 169
9.16 The printer Example - Illusrative Enrichment with a Reparation Pool . . . 170
9.17 CR6 - Policy Model . . . 171
9.18 CR - Violation Model . . . 171
9.19 Expense Refund Process - in xBPMN . . . 172
9.20 CR7 - Policy Model . . . 174
9.21 CR7 - Violation Model . . . 174
9.22 CR8 - Policy Model . . . 175
9.23 CR8 - Violation Model . . . 176
9.24 CR8 - Illustration of the *subjectCommittedPriorViolation(Subject, ASE)* predicate 176
9.25 CR11 - Policy Model . . . 177
9.26 CR11 & CR12 - Policy Models . . . 179
9.27 Informal Proposal to Model Article §164.512(f) of HIPAA . . . 181

10.1 CoReL: A DSL or a Semantic Bridge? . . . 188
10.2 Resolving the RCM Semantic Divide . . . 191

List of Tables

1.1 Examples for Types of Regulatory Compliance Requirements 8
1.2 Coarse Evaluation of Compliance Management Strategies 8
1.3 Research Questions 11
1.4 Contributions Answering the Research Questions from Table 1.3 12

3.1 Evaluation Notation of RCM Solutions 46
3.2 Comparison of RCM Solutions along RCM Framework Evaluation Criteria 48

5.1 Distribution of Approaches Along the 2 Dimensions 92
5.2 Compliance Requirement Examples 94
5.3 Compliance Requirement Examples 106

7.1 CoReL Symbols 126

9.1 Compliance Requirements - Printer Example 160
9.2 Compliance Requirements - Expense Refund Example 173

Part I

Domain Analysis

Chapter 1

Introduction

Brick Walls are there for a reason; they let us know how badly we want things.

Randy Pausch.

Compliance management has received considerable attention in recent years. It was shown to be a very challenging multi-disciplinary problem. This is especially the case in the context of business processes and more generally, process-centered enterprise information systems. Business Processes (BPs) count as one of the most important assets of companies. BPs describe the behavior of the enterprise, and represent its dynamics (e.g., how resources are moved around and to which purpose). Ensuring the compliance of processes to legal regulations, governance guidelines, and strategic business requirements is a sine qua non condition to controlling what happens both inside and on the outside interfaces of the enterprise. Implementing business process compliance requires means for modeling and enforcing compliance measures. In this work, we will provide a broad definition and an analysis of the problem of *enterprise regulatory compliance*. We introduce business policies as a fundamental concept for holistic management of enterprise compliance. We explain the usefulness of automation in compliance management and present the dissertation thesis.

> This thesis states that it is feasible and advantageous to use business policies to achieve the vision of:
>
> - Business user-friendly modeling of compliance regulations using visual-textual compliance decision-making modeling.
> - Enabling automated checking of certain sub-classes of these regulations on business process-centered enterprise models.

We present a holistic framework for managing business policies and assisting business users in using them. We describe a theoretical framework for formally modeling Compliance Requiremens (CRs) and study properties of the compliance problem. We then show how two different types of compliance checking may be conducted using business policies. We illustrate the feasibility of our approach on a case study. We eventually proceed to a discussion of the soundness and practicability of our approach.

1.1 Context

Compliance management has received considerable attention in recent years in both industry and in research [Kha12]. In the industry, awareness grew dramatically after several scandals starting in 2001 and the ENRON fraud which was directly correlated with the advent of the Sarbanes-Oxley Act (SOx, 2002) [EKP09]. Markets now shifted towards a tighter regulatory control of economic

activities, through a wide variety of regulations to be implemented by organizations (e.g., drugs and food, healthcare, finance, privacy). This fact has become more evident after the last economic crisis of 2008. The cost of achieving regulatory security compliance for example is on average 3.5$ million each year per company in the U.S.A., according to a survey of 160 individuals leading the IT, privacy and audit efforts at 46 multinational organizations carried out in 2011 [Wor11].

A research conducted by the Ponemon Institute[1] showed that the average cost for organizations that experience non-compliance is even higher at 9.4$ million. Costs originate for a variety of reasons such as *'business disruption, reduced productivity, fees, penalties and other legal settlement costs'* [Wor11].

Compliance is especially important in the context of business processes (BPs) [KMKP11a]. After analyzing industrial solutions to the management of regulatory compliance, we distinguish two main approaches: (i) compliance audits, and (ii) software implementations. While both solution categories suffer from high costs related to the external expertise that must be acquired by the enterprise, they present different disadvantages and advantages, analyzed in [EKP09].

Compliance audits are hard to automate since they require human intervention. Moreover, audits are error-prone and do not cover the whole enterprise model as they are conducted on samples of process logs or on selected parts of the information system. Software approaches are inflexible and are not generic, meaning they are usually specifically targeted at one compliance problem and hard to be reused for other types of problems.

Additionally, both approaches are so-called reactive approaches, since they do not enable early discovery and handling of situations eventually leading to violations before the latter happen [SG10]. Finally, it is a challenge to find adequate solutions supporting the full regulatory compliance lifecycle shown in Figure 1.1 (see the right side), as the mechanisms needed for each phase of the lifecycle are different. For instance, many approaches only tackle verification (i.e., static checking of compliance models) and do not tackle monitoring (i.e., dynamic checking of compliance models). On Figure 1.1, the dashed lines show that some phases of the BPM and RCM life-cycles must be aligned.

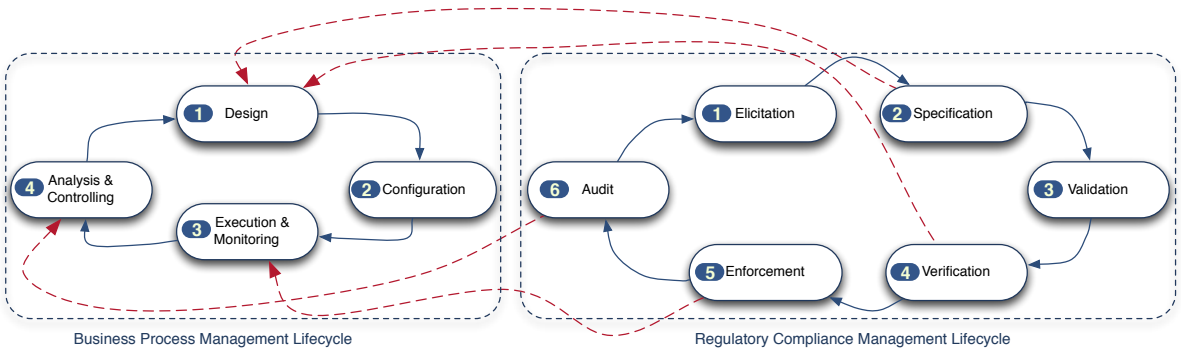


Figure 1.1: The lifecycle of regulatory compliance management

[1] http://www.ponemon.org/

1.2 Compliance in the Context of Business Process Management

Business Process Management (BPM) is the discipline of capturing, modeling, implementing, and controlling all activities taking place in an environment defining the enterprise, and this, in an integrated manner [Sch00a, Sch99]. BPM is an incredibly rich discipline and is the meeting point of several communities: information systems, computer science, business management. Several definitions, methodologies, languages, frameworks, and tools that support one or many of the listed aspects have been proposed [AH02, MW09, Wes07, zM04].

Business processes support and realize value-adding activities inside companies. These BPs may span several functional silos (e.g., departments, branches) in the enterprise's organization. As the enterprise as a whole is subject to various types of regulations, it becomes thus necessary to provide means of ensuring that a whole process is compliant with a given regulation. Compliance management is the term referring to the *methods and frameworks used to document compliance requirements and ensure that the enterprise conducts its activities accordingly.* It is worthwhile emphasizing the meaning of regulation here, which is not limited to legislation (i.e., laws) but covers instead any authoritative set of rules or preferences dealing with a procedures, guidelines or obligations. Non-compliance with a regulation diminishes the added-value that a business process provides for the organization. The reason for this may be non-optimal alignment with (i) regulatory obligations (ii) quality standards, (iii) partner (i.e., client, supplier, channel) service agreements. Non-identified security flaws may cause harm by threatening the availability or integrity of business processes.

Based on our current understanding of the term *compliance*, we give a high-level definition of Regulatory Compliance Management[2](RCM) as follows.

Definition 1.2.1 (Regulatory Compliance Management (RCM))**.**
RCM is the term referring to the use of means such as artifacts (e.g., rules, controls) and procedures (e.g., policy handbooks, processes, audits) to avoid or adequately react to undesired states of the enterprise or to undesired events occurring within the enterprise. RCM covers all business domains (e.g. health, finance, telecom), functional domains (e.g., security, quality, privacy) and regulatory domains (legislation, contracts, norms, standards).

Compliance management is the discipline of ensuring that enterprises behave according to a given set of prescriptions specified in a regulation. Usually, when speaking of regulatory compliance, these constraints on enterprise behavior are expressed as regulations, i.e., documents such as laws (SOx Act [otUS02]), contracts between companies, norms and standards (ISO 27000x[3]) or any other form of internal or external guidance (e.g., internal risk management directive documents).

Non-compliance to regulations can also be the cause of judiciary pursuits as many financial scandals in recent years have shown. Another repercussion to violating a regulation may be a loss in terms of image or loss of business opportunities. This has happened in the US and in Europe as the examples from various industries such as the telecom, consumer goods, banking, or automotive have shown (e.g., Enron [Lev02], Barclays[4], Worldcom, Societe Generale,

[2]from this point onwards, we may simply refer to it as compliance management or RCM
[3]http://www.27000.org/
[4]http://www.businessweek.com/articles/2012-07-03/the-libor-scandal-claims-its-first-ceo

Roche, Siemens, Volkswagen or Parmalat[5]). It is partly due to these scandals that laws and legal guidelines have been designed in order to protect companies and their stakeholders from violations such as manipulations of financial reporting data [RCJL06]. Two major examples are the Sarbanes-Oxley act[6] and Basel III[7].

The Sarbanes-Oxley act [8] [otUS02] was passed in 2002 after the Enron financial scandal [9]. SOx is one of the legislations that generated most reactions in the form of numerous publications on compliance [AIS09]. This is because companies doing business in the US are obliged to implement it and because of SOx's high implementation costs [TFL07]. Sections 302 (Certification of Disclosure in Companies' Quarterly and Annual Reports) and 404 (Definition of Internal Controls over Financial Reporting) of SOX are IT-related and thus explain the interest of the computer science community in both sections [RCJL06]. The European resp. Japanese equivalent to SOx is called EuroSOx[10] resp. JSOx and was passed by the European commission in 2006. BASEL-III[11] is a proposal by the Basel committee on banking supervision[12] that seeks to align regulatory capital requirements with operational and credit risk [CCHP04]. As an example of a paramount regulation in a different domain, ISO/IEC 2700x [Org12] is the internationally recognized standard for IT information security and may be used as a reference when defining enterprise security guidelines.

By extension, RCM also refers to standards, frameworks, and software used to ensure the company's observance of regulatory texts. In the context of BPM, compliance management applies to business processes and related resources like data and systems. As we said earlier, strong regulatory compliance management has both short-term advantages such as cost savings, reduced governance complexity, as well as long-term consequences like avoiding judiciary pursuits, image protection or enhanced market trust. The main challenge in dealing with compliance is that of achieving automation. The benefits of automation are numerous, and are best highlighted when observing how compliance initiatives are conducted. We can distinguish between two major types of compliance initiatives: (i) compliance audits and (ii) software implementations, which we briefly overview in the next section.

1.2.1 Shortcomings of Regulatory Compliance Management Methods

For many regulations, looking for means of automating the compliance audit process, as well as increasing its coverage of the organization and processes represents a real opportunity for substantial enhancement of its governance while decreasing costs. We look at the two major types of implementing RCM in enterprises, which are manual audits and software implementations.

1.2.1.1 Compliance Audits

Compliance audits are conducted by analysts who combine expertise in the regulation of interest and in the organization and business processes of the enterprise. Alternatively compliance audit teams are formed out of experts in both aspects. Typically, auditors look at a sample of the enterprise's activities for a limited amount of time, from a day to several weeks, or check past activities (e.g., process logs). The auditors then have to ensure that the requirements and guidelines specified by the regulation are kept to and

[5] https://en.wikipedia.org/wiki/Parmalat
[6] http://www.soxlaw.com/
[7] http://www.bis.org/bcbs/basel3.htm
[8] The Public Company Accounting Reform and Investor Protection Act is also known as the Sarbanes-Oxley act
[9] SOx is the usual abbreviation for this act.
[10] http://www.eurosox.dk/template.php?tid=304
[11] http://www.bis.org/bcbs/basel3.htm
[12] http://www.bis.org/bcbs/about.htm

implemented as specified. Based on their observations the auditors produce a report assessing and justifying the degree of compliance of the enterprise to the regulation. The auditors often provide recommendations to guide the enterprise's efforts in reaching a higher degree of compliance for the next audit. Such audits are expensive and are usually conducted once a year.

This method of compliance checking has some advantages for the enterprise, but also some inconveniences. First of all, the advice given by experts is surely to be expected to be very valuable, since these experts accumulate years of experience with the regulation and understand its implications for the enterprise's processes. Hence, it seems reasonable to assume that their advice is to be very accurate and to pin-point the critical areas of high risk and highest optimization potential. However, these experts are humans and because only a sample of an enterprise's processes is looked at (for obvious size reasons), experts are likely to overlook both violations as well as areas of the enterprise which are non-compliant. Audits also require manual labor and are expensive both financially and in terms of time. Additionally, enterprises trust the auditors based on their oath, but deep knowledge about the regulation and the opportunities for accelerating the pace of strengthening its adherence to the regulation stays outside of the enterprise. This may slow down the enterprise in its acquisition of regulatory knowledge which constitutes a competitive advantage.

Compliance audits present the advantage of relying on intangible human knowledge in the form of the experience of the auditors. This experience allows to combine knowledge of a number of different regulations and in a great number of contexts. This experience may allow more adequate selection of processes and efficient testing of the processes against specific elements of a given regulation. Recommendations may even be given based on best practices and knowledge of previous handling of the uncovered problems.

However, uncovering the root causes of the violations will most of the time be left up to the owners of the business processes. This method also does not scale with the number of the business processes and the relevance of the audit results may be regarded by compliance stakeholders as decreasing if the processes covered are a tiny percentage of all business processes.

1.2.1.2 Compliance Software

Another approach which has appeared in recent years is custom software for supporting compliance tasks. This software typically tackles a specific regulation or heavily recurring compliance principles such as segregation of duty (SoD). Segregation of duty specifies the set of (or constraints on the set of) roles or rights that an individual may cumulate depending on the activities concerned. For instance, a department head may decide on budget allocation for company gifts but may not validate this budget; only his superior may do that. A symmetrical way of viewing SoD is to define what roles are jointly required in order for an individual to carry on a process or a task. Such principles must often be instantiated on a large scale and must be instantiated for a huge number of individuals, roles and activities. This makes software implementation an imperative to assist with this task.

While software offers great support for automating the checking or enforcement of often large scale recurring rules, this approach suffers from some drawbacks. These systems cost money and require the heavy involvement of the IT department in the interpretation and implementation of compliance requirements. The integration of each system with the business processes also induces additional costs. Moreover, these systems work each for a specific set of rules of regulation,

making using a unified software for treating all kinds of regulations the enterprise needs to comply to a challenging task. This task is even more complicated for the controlling department which must report on the compliance to each regulation using various reports from various systems. With genericity being hard to fulfill, a holistic (covering all of the enterprise's processes) and unified (usable for all kinds of regulations) system may seem out of reach of most companies.

Table 1.1: Examples for Types of Regulatory Compliance Requirements

Domain	Abstract Example of a Regulatory Compliance Requirement
Banking	Rules managing the conditions under which financial transactions may be executed.
Banking	Documents that may be accessed by transactions.
Healthcare	Drug administration rules.
Healthcare	Patient form fill-up obligations.
Security	Identity protection constraints.
E-Commerce	Commitments to service level agreements between provider and contractor of a service.

The methods based on software may scale better with a greater number of business processes in terms of execution speed. Nevertheless, greater efforts are needed to deal with several regulations different in nature and covering distinct domains as shown in Table 1.1. Software implementations are not very flexible as separate sections of a regulation that require different checking methods may complicate the software and hinder efficient execution. Moreover, every change made to the regulation or to the business processes will require extensive testing to ensure the compliance software is efficient. We also observe that this leads to a high degree of specialization in existing compliance software.

1.2.1.3 Coarse Evaluation of Audit- and Software-Based Solutions to RCM

Table 1.2 summarizes our coarse evaluation of the main challenges facing enterprises in enhancing the practice of enterprise regulatory compliance management. We distinguish four main challenges: unification, holism, expressiveness, automation. The meaning of each of these criteria is explained in the table. These four criteria are applied to the two main application techniques of compliance (audits and software-based). The check resp. cross symbol means that the criterion is satisfied resp. not satisfied by the technique.

Table 1.2: Coarse Evaluation of Compliance Management Strategies

Criterion	Meaning	Audit	Software	Ideally
Unification	coverage of multiple regulations domains (e.g., financial, drugs, health, security, etc.)	✔	✗	✔
Expressiveness	coverage of multiple types of constraints in a regulation	✗	✗	✔
Holistic	coverage of multiple types of enterprise artifacts	✗	✗	✔
Automatic	coverage of multiple regulations	✗	✔	✔

Unification means to be able to deal in the same manner (i.e., using the same language or

framework) with several regulations different in nature. Difference among regulation manifests in the domain they cover, the type of constraints they place on the enterprise, and their impact on the enterprise model. This criterion is related to *expressiveness*, by which we mean the support for multiple types of constraints placed on enterprise artifacts (including processes). Constraints are different depending on whether they place structural, temporal, contractual or other kinds of constraints on the enterprise model [KMKP11b]. Usually, logics which are more and more powerful and expressive must be designed to deal with this criterion.

Holism refers the ability to consider any type of enterprise artifacts (sometimes referred to in the thesis as enterprise elements) for the checking of compliance. Some approaches to compliance deal with sets of business roles and authorizations, others with certain types of resource usage or only process tasks. However, proposing a generic approach to model compliance imposes to consider them indiscriminately. *Automation* is in our biased opinion the main criterion of interest for companies. If a company can automate checking of compliance requirements over large parts of its enterprise model (e.g., large numbers of its business processes), then it can certainly achieve a considerable win in terms of money and time saved, as well as lowered risk linked to possible faults or undesirable behavior in the processes.

1.3 Thesis Design

In this section we describe the scope and objectives of this research work. This thesis is placed in the context of business process-centered enterprise modeling, in which business processes are the central component. We bluntly say that business processes are the dynamic part of the enterprise model, connecting various elements of its architecture to achieve business goals. We concentrate on a setting where the motivations of the users of a solution to RCM is to model an organization and its activities, while making sure regulatory constraints are indeed taken into account.

1.3.1 Thesis Scope

In Figure 1.1, both the BPM (left part of the figure) and the RCM lifecycles (right part of the figure) as well as their interdependencies are shown. The simplified BPM lifecycle [Wes07, AH02, zM04] is divided into four phases of design (modeling of business processes), configuration (parameterization, data and service bindings, etc.), execution and monitoring and finally analysis (e.g., performance, quality, etc.) and controlling of business processes after execution.

In the right part of Figure 1.1, the general RCM lifecycle consists of six phases. After a first phase of elicitation where (i) the sourcing and scoping of the applicable regulations and then (ii) the regulations which have to be processed by experts and interpreted adequately for the organization, the compliance requirements included in the regulation are specified in a second phase called specification, for example as models. These specifications of compliance requirements must refer to BP models they are defined for. This is usually needed for documentation and analysis (e.g., traceability, impact analysis, etc.). This is represented by the dashed line between the step 2 of the RCM lifecycle and step 1 one the BPM lifecycle.

The third phase covers the validation of these compliance requirements, for removing and resolving inconsistencies or incompleteness. The fourth phase tackles the static checking of the

specification of the compliance requirements on the concerned enterprise models such as business process models. There is a dependency between this phase and phase 1 of the BPM lifecycle since verification must be realized on BP models.

Many compliance requirements can only be checked during execution of business processes as much information required for checking is only available at runtime. This is covered by another step which is the enforcement of compliance requirements during execution of enterprise models, which has a dependency to phase 3 of the BPM lifecycle as it must be integrated with the mechanisms for executing and monitoring business processes. The last step consists of auditing the logs of executed business processes for conformance to compliance requirements. As for previous phases, the Audit phase is dependent on phase 4 of the BPM lifecycle.

In conclusion, our work is designed to support RCM in interfacing with the BPM lifecycle. The research conducted in this thesis must consider the research questions and research results from a business process-centered enterprise modeling perspective. We must also evaluate the implications of our research on these process-centered enterprise models.

1.3.2 Thesis Objectives

Many attempts have been made in research at providing both usable and tractable, as well as powerful and expressive solutions for supporting compliance initiatives in BPs. However, most of these solutions focus on the challenge of formally describing constraints contained in compliance requirements in order to automatically verify them [Kha12]. The focus is not put on real end users of compliance frameworks, who are business-users and not computer scientists, making for low acceptance of novel solutions from research.

We aim first at analyzing the requirements for a framework that provides support for all the identified phases of the RCM lifecycle. This analysis is also based on the research realized in other works tackling RCM. Our second objective is to tackle this issue and provide an adequate paradigm for regulatory compliance management (RCM) that is suitable to business users. Moreover, compliance requirements do not only place constraints on enterprise models, they describe decision-making preferred, allowed or compulsory behaviors. This decision-making takes into account the context and other properties of the current state of business. Implementing these compliance requirements requires means to model these alternative behaviors and the decision-making associated with them.

In order to do that, we study the semantic gap between the conceptual domain in which business users think about compliance and the ones needed for a formal representation of compliance. A new modeling language is needed for creating reusable, lower-complexity and business-user friendlier compliance models. One very important criterion for this is that the language semantics support both automatic verification as well as enforcement of compliance requirements at the push of a button. An additional objective in this thesis is to study how model-driven engineering can be leveraged to support RCM.

1.3.3 Research Questions

We reckon through analyses of compliance management scenarios and existing research on the topic that automation, pro-active compliance management as well as support of the full BPM/RCM lifecycle are valuable capabilities that an RCM framework should provide

[SASI10, Kha12]. Hence, our research seeks to achieve some of the previous three capabilities by answering the research questions listed in Table 1.3.

Table 1.3: Research Questions

ID	Research Question
	Modeling
RQ 1	How to make RCM Modeling more amenable to Business Users?
RQ 2	How to exploit the power of logical formalisms for compliance management?
RQ 3	How to cover several types of enterprise business artifacts?
RQ 4	How to decouple compliance modeling from EM/BPM notations?
	Checking
RQ 5	How to realize *'verification'* using compliance models?
RQ 6	How to realize *'enforcement'* using compliance models?
	RCM Language Engineering
RQ 7	How far can the Model-Driven Engineering discipline support our endeavour?

1.3.4 Results

In this thesis, we develop the ***ASE (Action, Subject, Entity) paradigm*** for formalizing behavior in enterprise models and link this behavior to enterprise artifacts. We also propose another approach for ***modeling the decision-making elements of compliance requirements***. This approach is based on separating the decision making layer from the constraint/rule (decision implementation) layer. Following the identification of a useful modeling paradigm, we derive the core components of the framework: (i) a ***method***, (ii) a ***language***, a (iii) ***semantics***, and (iv) ***tool support***. The contributions of this thesis are listed according to the research question they contribute to solving in Table 1.4.

In order to achieve this vision, we define a ***visual Domain-Specific Language (DSL) for decision-making*** as a core contribution. This language is named the ***Compliance Representation Language (CoReL)***. The rest of the thesis is about ***integrating this DSL into enterprise modeling*** and ***using this DSL for compliance analysis***.

CoReL is based on the core concept of *business policy*, which we propose to fill the semantic gap between business users and formal methods. Business policies allow to model decision-making by breaking it into reusable parts [KMKP11a]. They allow to reason about all possible violation types and how to adequately react to each of them depending on a number of factors such as the situation. We then define formal operational semantics for CoReL [KMKP11a].

Most importantly, the semantics of CoReL proposed here are ***integrated with a formal execution semantics of a sub-class of business process models***. Also, we use ***model-driven engineering*** and in particular formal ***metamodeling*** and ***model transformations*** in order to show how CoReL may be ***integrated into a concrete business process modeling notation***.

This thesis goes several steps ***towards supporting an integrated BPM and RCM lifecycle***.

The following table gives a summary of contributions:

Table 1.4: Contributions Answering the Research Questions from Table 1.3

Research Question	Contribution	
	Modeling	
RQ 1	CoReL: Visual DSL.	Chapter 5
RQ 2	CoReL decoupling of decision-making (policies) and constraints/rules.	Chapter 5
RQ 3	ASE paradigm.	Chapter 4
RQ 4	ASE paradigm and its integration into CoReL.	Chapter 6
	Checking	
RQ 5	Formal operational semantics of CoReL.	Chapter 6
RQ 6	Formal operational semantics of CoReL.	Chapter 6
	RCM Language Engineering	
RQ 7	Use of metamodeling, model composition, and state-of-the art MDE technologies in building tool support.	Chapter 5

1.3.5 What This Thesis Is Not

In this thesis, we do not develop a new (business) rule modeling language, nor do we contribute anything related to the logics required to accurately represent and reason on constraints/rules.

1.3.6 What This Thesis Is

This thesis is a work on language engineering in the RCM domain for the purpose of modeling and verification.

1.4 Structure Of The Thesis

The remaining of this thesis is structured as follows. Chapter 2 presents the background required to understand the contributions of this thesis. We give a concise background on business process management (BPM), model-driven engineering (MDE) and on the formal modeling language and tool Alloy. Chapter 3 introduces the state of the art on compliance management as well as the most relevant related work, leading up to a framework to classify and compare compliance management approaches.

We propose a minimal formal model for representing behavioral descriptions of enterprise models we call system models in Chapter 4. Then, Chapters 5 and 6 introduce the CoReL

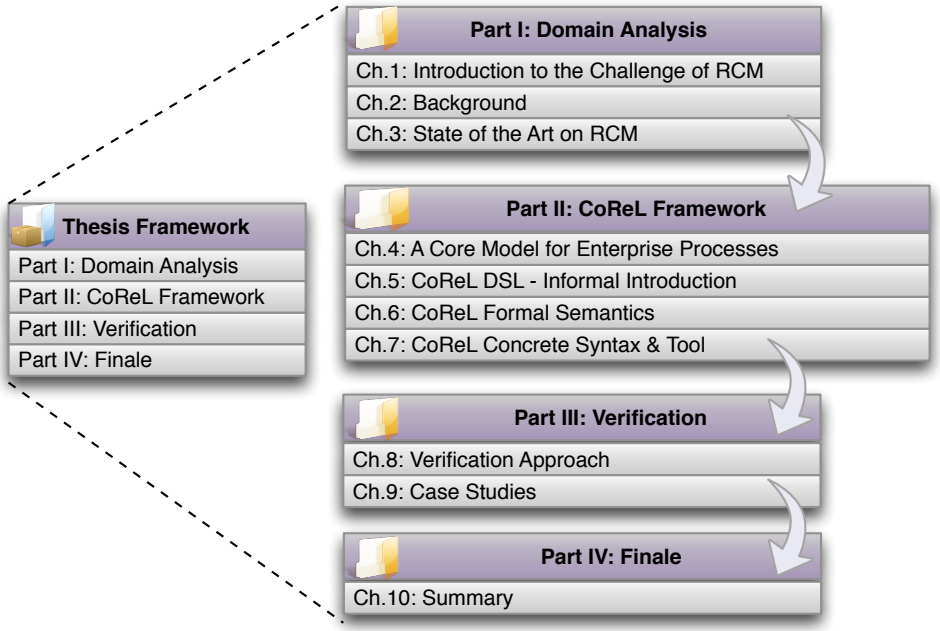


Figure 1.2: Thesis Structure

language and its semantic foundations. This is followed by Chapter 7, which describes the concrete syntax and the diagrams developed as part of the framework.

Consequently, Chapter 8 presents the verification approach we use to realize compliance checking, while this is examplified in Chapter 9 using several example-based case studies. Finally, the last Chapter 10 builds on the previous and provides perspectives and a discussion of the contributions and the results of the research.

1.5 Summary

Compliance to regulations is a critical issue in the governance of enterprise architectures and consequently of business process management. In practice, current solutions for compliance implementation rely heavily on expensive manpower or on large, rigid software implementations. On the other hand, logics and formal verification languages offer a powerful, potentially highly exhaustive and precise alternative to existing solutions.

However, using these formal languages to offer automated support to the full compliance lifecycle presents several challenges. The most important one is, next to enabling fully automated checking of compliance, making the formal foundations of the framework suitable for business users who do not have the required skills to directly deal with these formal foundations. The research realized in this thesis studies the use of business policies as fundamental decision-making elements to provide a small, yet expressive visual language for compliance modeling. Combined with the application of model-driven engineering techniques of composition and transformation, it shows the feasbility and advantages of applying model-driven engineering to the RCM problem.

Chapter 2

Background

The design of computing systems can only properly succeed if it is well-grounded in theory, and...the important concepts in a theory can only emerge through protracted exposure to application.

Robert Milner.

In this chapter, we will provide general background information and definitions, which are required to fully grasp the context and the contribution of this thesis. We do this for the topics of Model-Driven Engineering (MDE), Business Process Management (BPM) and for the Alloy formal method.

2.1 Business Process Management

2.1.1 Business Processes

According to Hammer in [HC03], a Business Process (BP) is a *"group of tasks that together create a result of value to a customer"*. Davenport [Dav93] sees BPs very similarly as *"a set of logically related tasks performed to achieve a defined business outcome for a particular customer or market"*. These two definitions come from management science.

In Weske's book [Wes07], another more technical view of BPs is taken. A BP is defined as: *"consisting of a set of activities that are performed in coordination in an organizational and technical environment [...] Each process is enacted by a different organization, but it may interact with processes performed by other organizations"*.

In another reference document, the workflow management coalition's (WfMC) terminology and glossary [Wor99], another definition is given. It is the *"set of one or more linked procedures or activities, which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships"*.

The last two definitions are closer to the use of BPs in this thesis. We view BPs as central enactment elements of any enterprise. BPs are its machinery, its muscle, the element of an enterprise that carries out change and transformation. Many of today's mainstream perspectives on BPM and its relevance in the industry come from the seminal book by Smith & Fingar [SF04]: "The Third Wave". The first wave of BPM was rooted in Taylor's theory of management [Tay11]. At this stage business processes were considered as immaterial and carried out manually by the workforce. In the 1990s during the BP re-engineering wave, BPs began to be supported and managed by information systems, particularly Enterprise Resource Planning (ERP) software.

The third wave however, began to see BPs as the central automation component of enterprises, destined to be implemented and governed for best performance.

We understand from the three different definitions that BPM is a complex topic tightly linked to the existence and operations of an enterprise. Business Process Management (BPM) [Sch99, Sch00a] is the discipline that deals with Business Processes (BPs). In [Sch99, Sch00a], BPM is defined as *"the discipline of capturing, modeling, implementing, and controlling all activities taking place in an environment defining the enterprise, and this, in an integrated manner"*.

2.1.2 Business Process Lifecycle

The BPM lifecycle describes the steps or phases linked to managing BPs. Figure 2.1 shows a slighltly modified version of the BPM lifecycle from a management perspective proposed in [zM04]. This lifecycle is quite complete in that in not only includes the steps of the lifecycle, but also the artifacts which are exchanged among steps.

Analysis The lifecycle starts with the analysis activity. The analysis is based on the process context (environment), which spans from strategic concerns (business goals, directives, etc.) to organizational structure. The output of this phase is a set of requirements for business process design, which include process objectives, key performance indicators, process owners, etc.

Design Based on the set of requirements from the previous activity, this activity is concerned with capturing either an existing (as-is) or desired (to-be) state of a process. Based on information originating from various sources, an abstract representation in a graphical notation is used to cover knowledge about business processes at hand[Wes07]. This includes defining process activities, specifying their order and constraints, assigning process roles and resources, etc. The output of this phase is a conceptual process model which integrates these different design perspectives.

Enactment Process enactment is the phase where the business process management system is actually deployed and executed. The BPMS infrastructure supports the execution of individual cases (process instances). During execution, different case data is recorded such as execution time, process branching decisions, resource access, etc. This data is used as input for the following two management activities.

Monitoring This activity is performed continuously for each case based on the assigned process metrics, such as maximum delay time. If a metrics threshold is reached, appropriate program logic is triggered in order to deal with the situation. This activity is essential in assuring the correctness of individual process executions.

Evaluation The evaluation activity is concerned with the aggregated view on individual process executions. Here, the process execution data is compared to the original design requirements of the process. The goal is to assess further potential process improvements and thus identify new requirements for process design. These new requirements are taken as input for the process redesign in the next iteration of the lifecycle.

Implementation This activity uses the conceptual process model as input. Implementation means the selection and configuration of IT artifacts which implement process activities, according to process control- and data-flow. During implementation, it may be discovered

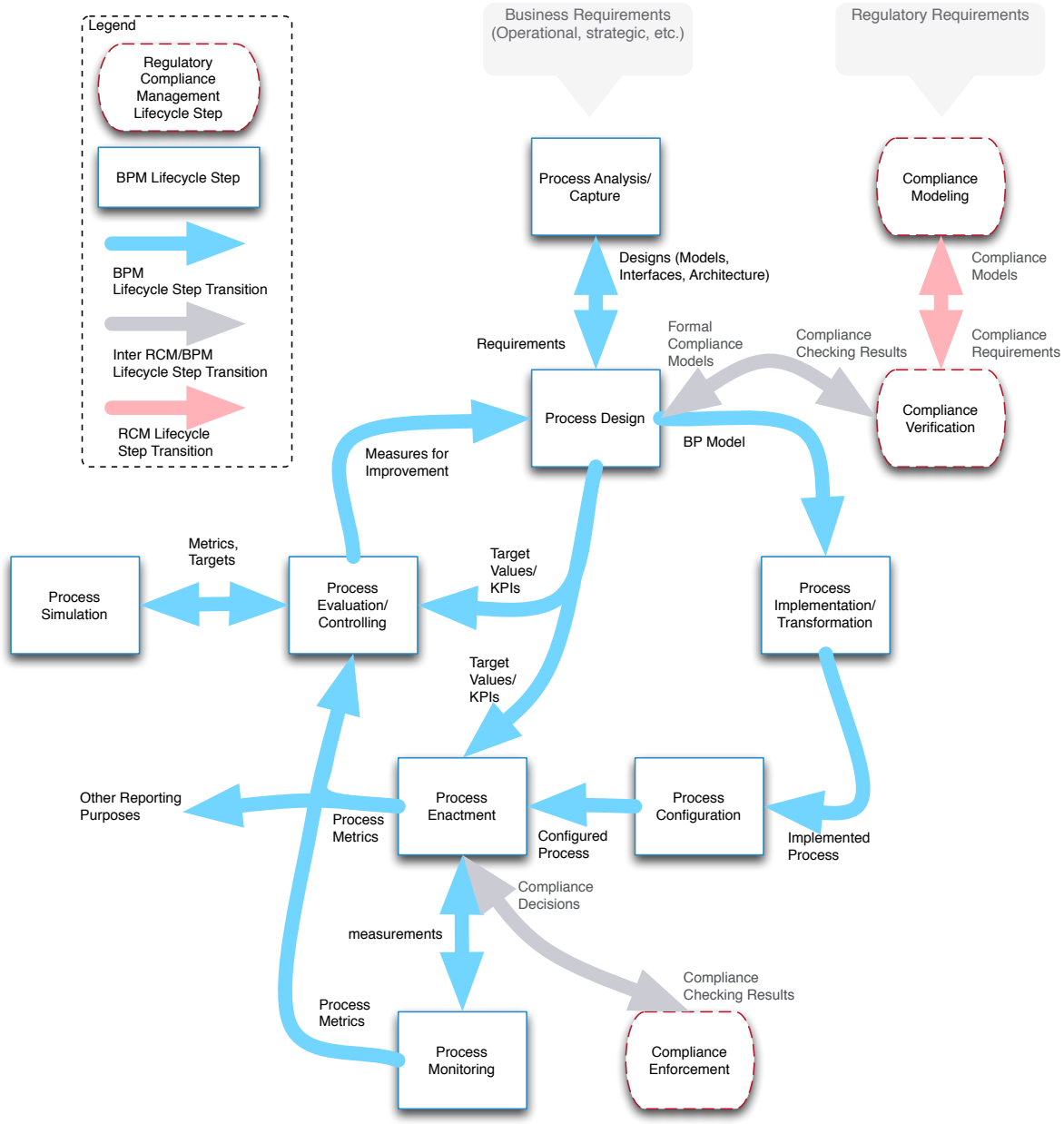


Figure 2.1: BPM Lifecycle - Adapted from [zM04] - Extended with RCM Steps Contributed in this Thesis

that the process can not be implemented as depicted by the process model. This provides a feedback loop to the design activity in order to perform required modifications.

The BPM lifecyle in Figure 2.1 has been extended with three steps, distinguishable from the BPM steps by using oval shapes, belonging to the RCM lifecycle. We do not show the full RCM here, as only some steps are relevant for this thesis. We explain these additional steps here.

Compliance Modeling is the step where compliance requirements contained in regulations are modelled in a specific language. This is to allow reusing these compliance models as artifacts for tasks such as verification and enforcement.

Compliance Verification is the step where compliance models are checked against business process designs, i.e., statically.

Compliance Enforcement is the step where compliance models are checked during the enactment step, against business process instances being executed, i.e., dynamically.

2.1.3 (Enterprise) Business Process Modeling

In order to describe a business process, several forms of process concepts need to be integrated in a business process model. The authors in [CKO92] provide a popular categorization of concepts that a process model should capture. They argue that information in a business process model should provide answers to the following questions: what is going to be done, who is going to do it, when and where it will be done, how and why it will be done, and who is dependent on it being done [CKO92]. These questions helped to group the commonly used process concepts into four perspectives: functional, behavioral, organizational, and informational process perspective. Many enterprise modeling/architecture methods or languages consider additional perspectives to these last four [vdAtHKB03, ea09b, DH08, Hoo09, BMS10, Mar10].

An example of a widely accepted method to capture different process perspectives is the so called ARIS House of Business Engineering [Sch99]. ARIS distinguishes between four perspectives: Organization (who), Data (what), Function (how), Product/Service (output), centrally connected by a process flow (in which order) [KNS92, DB07]. Recent research on process perspectives [Wes07] also reinforces the ideas outlined in [KNS92, CKO92] [CKO92], where control flow, function, data, and organization modeling are seen as the core ingredients of a business process model.

Control Flow perspective (or process) perspective describes activities (tasks) and their execution ordering through different constructors, which permit flow of execution control.

Data perspective deals with business and processing data. Business documents and other objects which flow between tasks, and local variables of the process, qualify in effect pre- and post-conditions of tasks execution.

Resource perspective provides an organisational structure anchor to the process model in the form of involved humans and device roles responsible/needed for executing tasks.

Functional/Operational perspective describes the elementary actions executed by tasks, where the actions map into underlying applications. Typically, (references to) business and process data are passed into and out of applications such as Web Services, allowing data manipulation within applications.

2.1.4 Business Process Modeling Languages

Process modeling languages [LLK09] can be classified into three main categories: informal, semi-formal, and formal languages [Lin08]. Informal languages are natural languages, mostly textual descriptions similar to plain english (or any other human written idiom) which may follow certain conventions, format, templates, and structures. These are the languages mostly used during the analysis phase where business processes are initially collected and documented. Semi-formal languages are graphical languages which introduce a set of notations with semantics defined in the underlying metamodels. Formal languages have rigorous and precise semantics defined in formal logics or mathematics. The work in [Lin08] compares most widely used process modeling languages such as Petri nets [Pet62], BPMN [OMG11], EPC [KNS92], which we will shortly describe.

Petri nets were introduced in the 1960s as a formal, graphical language for modeling distributed systems [Pet62]. Due to their well defined mathematical properties, they are widely used in workflow management and software design in practice. Petri nets are also still popular in the BPM research community, where various extensions regarding complementary process modeling perspectives have been proposed [vDdMV+05, Wes07, vtKB03].

Event-driven process chains (EPC) [KNS92] were created in the 1990s as part of the ARIS framework. In its basic form, the EPC concepts bear close similarities to Petri nets, as they support modeling of processes as a chain of events, triggering a function which in turn again results in events. There are various extensions of the basic EPC concepts, which provide support for modeling the organizational and informational process perspectives. EPC is a graphical modeling language which is targeted to be easily understood and used by the business people.

The Business Process Modeling Notation (BPMN) [OMG11] is a new standard for modeling business processes. BPMN belongs to the category of semi-formal modeling languages, intended to support business process management for both business and technical users. BPMN roots stem from the activity diagrams in UML [Obj11] and it is gaining increasing adoption in industry and academia. At the time of this writing, BPMN 2.0 version of the standard is being finalized. This new version includes extensions to the notation (e.g. for expressing collaboration between process participants), metamodel specification, and interchange format.

If we observe the historical development of the above mentioned modeling languages, it can be noted that over time they got richer in expressiveness (scope). In the case of EPC, if we compare the notation in [DB07] with the initial version in [KNS92], we notice that the trend of notation extensions aims to support a broader business context. For example, the extended EPC described in [STA05] and [DB07] allows for modeling of business objectives, key performance indicators, risks, and internal controls in the context of business processes. However, the proposed extensions i) remain underspecified with respect to their semantics, ii) are not well integrated to the basic EPC metamodel, and iii) have weak links to other extensions. In the case of BPMN, on the one hand it has been extended to allow for modeling of B2B collaboration, whereas on the other hand a formal semantics has been proposed for the behavioral process perspective [OMG11].

Extensions on the business process notation level in general fall short of seeing processes in the overall business context. For example, one of the major issues in BPM identified by the authors in [BIS+07] is the "broken link between BPM efforts and organizational strategy" [BIS+07].

Detailed descriptions of different process modeling languages can be found in

[DvdAtH05, Wes07]. In the next section, we focus on the second aspect important for our work which will turn instrumental in providing semantics for our modeling languages.

2.2 Model-Driven Engineering

Model-Driven Engineering (MDE) [Ken02, Sch06, CATK03, FR07] is a novel approach to Software Development. It elects models as first-class artifacts during all phases of Software Development. The goal is to improve and maximize the productivity in both short and long term. In a short term, by increasing the average functionalities a software system is delivered with; in a long term, by reducing the costs of maintenance and accuracy of the systems by reducing their sensitivity to change (coming either from technologies, or from stakeholders) [CATK03].

From a methodological point of view, MDE uses models throughout the development of complex systems to improve the effectiveness of every days tasks, such as documentation of the system at early stages, specification of the different components of the system, design of the overall architecture, development itself, versioning, maintenance, etc. From an engineering point of view, models are used to provide a team involved in the development process the most accurate level of abstraction to deal with the essential complexity of each task and let all the machinery behind the tool support deal with (most of) the accidental complexity [Bro86], by using transformations to manipulate models.

2.2.1 The Need for Abstraction

Since computer scientists have begun to program computers, they faced the need to increase the level of abstraction in the expression of programs. The history of programming languages, the first interface for interacting with computers, has consistently offered new paradigms for writing and designing programs.

The first generation programming languages were operating at the level of the CPU, directly manipulating the operation codes used by a particular processor; second generation languages, although specific to a particular architecture, offered a first layer of abstraction by manipulating mnemonics and macros, allowing programmers to abstract from the internal binary representation of instructions that was clearly not human-readable and very error-prone; third generation languages, with their large scale of abstraction levels and programming paradigms, offered to programmers the possibility to specify what the machine should do rather than how.

The raising of programming abstraction levels has been continuing since then, and today's object-orientation is considered to enable programmers to tackle problems that were impossible a few decades ago. Model-Driven Engineering and / or Development can be seen as the continuation of this effort to propose to programmers the best level of abstraction they need in order to specify, develop and maintain a system: instead of describing how a system is built, they can specify which functionalities are required to build the system. As such, it can be considered as the natural continuation of Object Orientation, and as a matter of fact, Object Orientation largely inspired the Model-Driven approaches; but in another way, MDE operates a radical rupture by trying to propose a dedicated modeling of every aspect of interest, going much deeper

in the way opened by Object Orientation [Com08].

Model-Driven Engineering (MDE) received a large acceptance at its beginning, and with the dissemination of the technique started the multiplication of approaches and model frameworks: Model-Driven Development (MDD) [CATK03, BCT05]; Model Engineering [JB06a]; Model-Based Approach (MBA) [AGERPS08]. One particular approach, Model-Driven Architecture (MDA) [Obj03, KWB03] is very often used (erroneously) as a synonym of MDE.

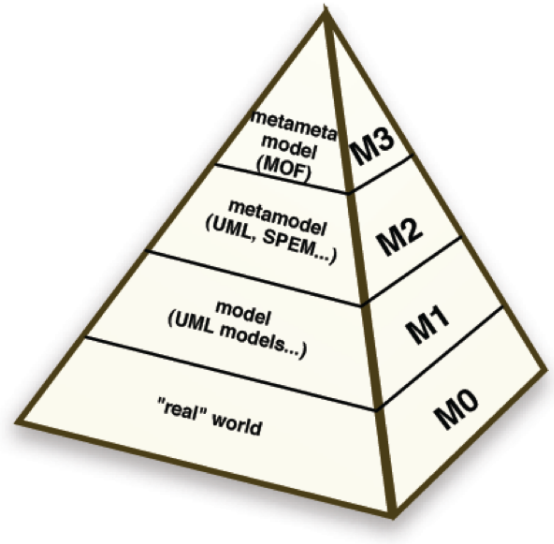


Figure 2.2: The OMG MDA Pyramid (from [tow07])

In Figure 2.2 we show the well-known OMG[1] pyramid. This pyramid shows four layers of abstraction, M0 to M3, where software artifacts reside. Artifacts populating the M0-level belongs to the real-life, i.e., in software engineering terms, objects in the memory of the platform executing the software. Artifacts in the M1-level are models of reality, e.g., a UML model describing a mobile phone application [KT08]. Models are widely used across all engineering disciplines and natural sciences, with different meanings, such as electrotechnical engineering, telecommunications, motors engineering, construction, mathematics, biology, physics, etc. Artifacts belonging to the M2-level are metamodels allowing the definition of these models. A metamodel [TK06, AK02, AK03] defines the set of models it accepts as instances. Metamodels can be used to partly define languages. Finally the unique artifact in the M3-level is MOF [Obj06, GRS09], the OMG/MDA language for metametamodeling, meta-circularly defined by itself.

2.2.2 The (Meta-) Modeling Approach

The MDE approach considers models and transformations as first-class entities, in order to specify and design applications and software systems at the right level of abstraction needed by the application domain the model is intended for.

In Computer Science, the notion of "model" covers a wide range of realities, depending on what specialization field one is talking of. For example in model-checking, the term model is

[1] www.omg.org

used either to designate the behavioral structure representing the system under study (basically, an automaton) or, more closely to mathematics, an interpretation of the property of interest expressed in a logics.

In the context of MDE, models are always language-based in nature and refer to artifacts formulated in a modeling language (among which UML is one of the most widely used) describing a system. Typically, a model is graph-based and visually rendered. The leitmotiv of MDE is the use of the "model" notion everywhere ("everything is a model" [JB06a]).

Models can be used at different purposes, sometimes overlapping: for documentation and / or communication purposes, where the model provides a more abstract way of explaining how a software and / or a system works, and to validate with stakeholders what was the intent of the system; for simulation and/or verification before the actual system is realized, to figure out in advance what are the issues and what optimizations can be operated, and to check in advance some properties of interest about the system.

2.2.3 Basic Ingredients for Metamodel-Based Language Engineering

2.2.3.1 Syntax: Abstract & Concrete

The syntax defines the concepts and relations a language or a model manipulates, together with the constraints concepts and relations are supposed to verify. In the context of traditional languages, the syntax is generally specified using a formal grammar (cf. for example, the Compiler Theory [ASU86]).When dealing with models, the syntax generally takes the form of a metamodel. But this syntax, generally qualified as "abstract", only captures the fundamental links between the elements described by the language; a more convenient way of manipulating them is needed and the language always comes with (at least) one "concrete" syntax, either textual or visual. To relate both syntaxes, a mapping, known as parsing [ASU86, JRAS97, LGKWNY08, dLV04, CDOP02], is needed to obtain the abstract representation in which all operations will be performed.

Furthemore, the abstract syntax expresses only structural dependencies between concepts via relations. There is always a need to constrain all the possible valid instances, either it is a phrase in a given grammar, or a model that conforms to a metamodel, because not all are structurally valid items are valid from the modeler point of view. This particular constraints are known as type-checking rules or static semantics when grammars are used; and simply constraints expressed in a dedicated language (among which OCL [JWAK03] is the most known). This particular relationship between instances and models is called constrained conformance [ABJM07].

2.2.3.2 Semantics

Semantics [HRNFN92, AK09, GRS10] captures the meaning of a formal (or natural) language, which could consist of defining the valid executions of the program. In fact, specifying a semantics consists in defining a semantic domain, reflecting how the concepts of the language are interpreted, and then providing a mapping from each concept to its semantic interpretation in the target semantic domain.

Having fixed a semantic domain, there are traditionally four styles [ZX04] for specifying the semantic mapping, depending what kind of formalism supports the specification:

Operational [Hen90] The meaning of the language constructs under definition is described by means of a state transition system, where the state basically embeds the semantic domain plus the machinery needed for the execution, and the transitions express rules explaining how the state evolves when executing a particular construct of the language. An example are Structured Operational Semantics (SOS)[Hen90].

Denotational [Win93, AFL99] The meaning of the language's constructs is given using denotations, i.e purely mathematical objects (like functions, functors and posets) called domains.

Axiomatic [Win93] The meaning of a language's constructs is described by means of logics stating properties (pre- and post-conditions, and invariants)which hold along the execution of the program.

Translational/Transformational Furthermore, there exists another technique, often called translational or transformational, where the semantic domain is constituted by another target language. This approach assumes that the target language acting as the semantic domain is well-known and/or well-understood, even if its own semantics is not mathematically defined. The ideal case is when the target language has formal semantics itself.

2.2.4 The Engineering of Metamodeling

Because in the MDE approach, we use models everywhere it is possible, formally defining a language through metamodeling uses three (meta-)models: one defining the abstract syntax, as it is classically done; one defining each concrete syntax, and one defining the semantic domain. In order to relate each model with the corresponding ones, MDE advocates the use of transformations [CH03, GLR+02, ES03, MJ02]. Figure 2.3 gives a characterization of these transformations in MDE.

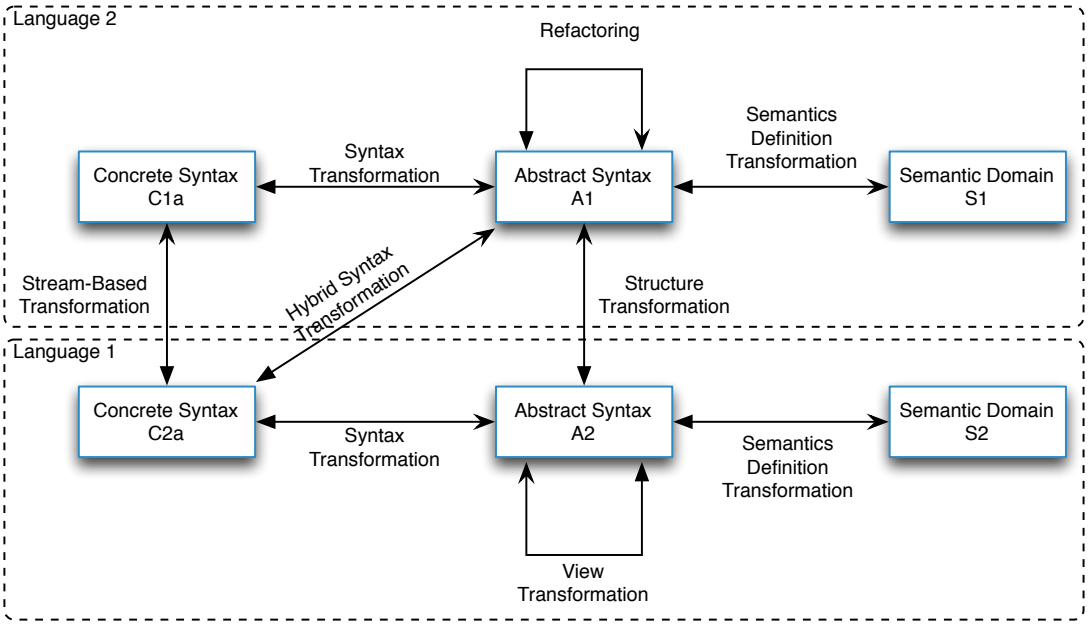


Figure 2.3: Model Transformations in Language Engineering (from [Kle08])

As such, a transformation is simply an operation between a source and a target model. If both source and target metamodels are the same, the transformation is endogenous, otherwise it

is exogenous [AK09]. From a semantic point of view, using endogenous transformation defines a simulation of the language defined by the metamodel, while using exogenous transformation defines a translation to a behaviorally equivalent metamodel.

2.2.4.1 Domain-Specific (Modeling) Languages (DS(M)L)

"A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain" [vDKV00].

Domain-Specific Languages [AK09] try to specify a solution to the development of a software directly at the level of abstraction domain experts deal with, by representing in the language the very same concepts used in the problem domain, and by achieving a high level of automation in the daily tasks the models are intended for: testing, validation, verification, simulation, and finally, production.
If languages provide high-level concepts, the development of the entire software solution for a given problem is always based on the knowledge of the experts to capture adequately the good concepts, and the past experience of the company the solution is developed for to reuse existing libraries and platforms. In this sense, the smaller the domain targeted by the Domain-Specific Language is, the better the automation can be.

DSLs trade generality for expressiveness in a limited domain. By providing notations and constructs tailored toward a particular application domain, they offer substantial gains in expressiveness and ease of use compared with General Programming Languages (GPLs, e.g., Java) for the domain in question, with corresponding gains in productivity and reduced maintenance costs. Also, by reducing the amount of domain and programming expertise needed, DSLs open up their application domain to a larger group of software developers compared to GPLs [MHS05].

2.2.4.2 Roles in Domain-Specific Engineering

In the design of a DSL, several roles involved can be distinguished :

Domain Experts are represented by persons having the knowledge about the problem domain: they manipulate on a daily basis the concepts that will be used in the DSL, are experts in the terminology, and are aware of the business rules that must be implemented as part of the behavior of the DSL. Depending on the domain of expertise the DSL, these experts can either be business practitionners or technical experts, or both.

Language developers (Metamodelers) specify the modeling language of the DSL. They work closely with domain experts to determine exactly which concepts and rules must be embedded inside the DSL, focus specifically on the range of the DSL in order to keep it as small as possible, and plan the future evolution of the DSL by evaluating new needs and integrating them. They formalise all the relevant concepts in a metamodel, and provide all the necessary tools for the models users: editors, documentation, etc.

Model Users (Modelers) use the DSL. Models can serve different purposes and be exploited by different people, depending on the particular usage a model is needed for: specification, communication, deployment, testing, documentation, etc.

The above roles are present for every DSL. But there are additional roles which are paramount to the success of a DSL solution.

Ergonomists are people who help metamodelers in order to have adequate and usable terminology, symbols and notations for concrete syntax, as well as ergonomic designs in the tools proposed to modelers. These people are not required, especially when using automated approaches that derive all the necessary tooling from the metamodel definitions, but they can help when the DSL solution targets a domain where interaction with non-programmer users is important.

Domain Framework Developers are experienced developers and application architects that develop the library code or component frameworks, and ensure integration between the framework and the code derived from the models. It is a crucial part since it contributes to the scalability and the quality of the DSL solution.

2.2.5 Formal Verification

The semantics of a program formalizes the set of all possible executions of a program in all possible execution environments. This semantics is usually an infinite mathematical object which is not computable: it is not possible to write a program able to represent and to compute all possible executions of any program in all its possible execution environments. Hence, in general, all non trivial questions about the semantics of a program are undecidable: it is not possible to write a program able to answer any question about the possible executions of any program.

Questions about a program are called properties and are also formally defined. Generally, (dynamic) properties belong to one of the two following categories: safety properties stating that nothing bad happens; and liveness properties stating that something good eventually happens. Properties can be seen as forbidden zones inside the a space defining the possible executions of the program [CC77].

If we focus on safety properties, their verification consists in proving that the intersection of the semantics of the program with the forbidden zone is empty. Since the program semantics is generally not computable, the verification problem is undecidable: it is not always possible to answer the safety questions completely automatically, with finite computer resources, without any uncertainty about the answer and without any human intervention.

Several techniques were developed over the years to overcome the decidability problem. testing/debugging consists in considering a subset of the possible executions; bounded model-checking consists in exploring the prefixes of the possible executions.

Static Analysis The abstract semantics is computed automatically thanks to predefined approximations [CC79, PC81, CC77], possibly manually parameterizable by the user.

Model-Checking The abstract semantics is provided manually by the user in the form of a finite model of the program execution (for example a finite automaton) [CC00]. In some cases the model can be computed automatically, by methods relevant to static analysis.

Deductive Methods The abstract semantics is specified by verification conditions and must be provided by the user in the form of inductive properties (true at each program step, such as loop invariants) satisfying these verification conditions [Cou02]. The inductive properties must be found manually by the user and the theorem prover sometimes needs assistance to prove that they are indeed inductive. To help the user in this discovery task, some of these inductive properties can be computed automatically, by techniques relevant to static analysis.

2.3 Alloy

Alloy is a so-called lightweight formal method. Alloy is a logic, a language and an analysis tool [Jac12], all at the same time. Alloy's logic combines the quantifiers of first order logic with relational calculus. Our introduction to Alloy is completely based on the reference book 'Software Abstractions' by Daniel Jackson [Jac12]. Alloy has been widely used for a formalization of various languages such as UML [SAB10], AspectUML [MV07], MOF [KM08], JAVA [DY13, Gal13, Rey10], concurrent systems [CRR09], and other systems (e.g., railway safety, file synchronisation and air-traffic control, healthcare systems [Jac13]).

In this section, a concise introduction to the Alloy approach is presented. A detailed presentation of the Alloy approach is available in the reference textbook on Alloy [Jac12]. After an initial introduction to the Alloy approach, we present the syntax and semantics of Alloy in 2.3.2. Then, we learn the function and usage of the Alloy Analyzer in 2.3.3. Finally, we give a small comparison of Alloy with other formal methods and motivate our choice of Alloy in this thesis in 2.3.4.

2.3.1 Introduction

The Alloy approach is based on three key elements: a logic (fundamental concepts), a language and an analysis. A basic knowledge of the Alloy language and the Analyzer is required for full understanding of this thesis. The logic introduced in the Alloy approach is a relational logic. It combines the quantifiers of first-order logic with the operators of the relational calculus. Fundamental concepts of the Alloy language are atoms and relations. Atoms are primitive entities which are indivisible (can't be broken into smaller parts) and immutable (their properties don't change over time).

Relations define relationships between atoms. They consists of a set of tuples, each tuple being a sequence of atoms. Relations in Alloy may be of any size and of arity greater or equal to one. A relation with no tuples is said to be empty. Unary relations with arity one are sets of atoms. Unary relations with a single tuple are scalars. Relations with arity two resp. three are said to be binary resp. ternary. A relation with arity of three or more is said to be a multi-relation.

2.3.2 Syntax and Semantics

The abstract syntax of the Alloy language is defined by a BNF grammar in [Jac12] in (p.261). In the following sub-sections, we introduce the main syntactical elements of Alloy: signatures and fields, operators and quantifiers, as well as facts, predicates and functions.

2.3.2.1 Signatures and Relational Navigation

A signature declaration defines a type as a set of atoms (i.e., a unary relation over atoms). A signature also defines a collection of relations called fields. An abstract signature has no atoms, but those of possible subsignatures (cf. definition of the ***extends*** keyword).

```
1 sig T1 {r:R}
2 sig T2 {s:R1→R2}
```

```
3 abstract sig T3 {}
```

Fields and Arities

The declaration above defines a type $T1$ and a binary relation r of range R and domain $T1$ ($r : T1 -> R$). The notation $a -> b$ where a and b are types defines an Alloy binary relation. Similarly, the second signature T2 defines a ternary relation between signatures $T2$, $R1$ and $R2 : T2 -> R1 -> R2$. We call $'r'$ a field of signature $T1$. This means that if some atom x has the type $T1$ then the expression $x.r$ will have the type R. More on the $'.'$ operator notation, called relational join, later. Multiplicities are assigned to a signature's fields:

```
4  sig T {r: one R}
5  // means that the binary relation relates every atom of T with a single atom of R.
6  //This is the default arity in Alloy and can therefore be omitted
7
8  sig T {r: lone R}
9  // means that the binary relation relates every atom of T with at most one atom of R
10
11 sig T {r: some R}
12 // means that the binary relation relates every atom of T with at least one atom of R
13
14 sig T {r: set R}
15 // means that the binary relation relates every atom of T with an
16 // (unordered and possibly empty) set of atoms of R
17
18 sig T {r: seq R}
19 // means that the binary relation relates every atom of T with an
20 // (ordered and possibly empty) sequence of atoms of R
```

Extends

Signatures may be subtyped by using the ***extends*** keyword. A signature declared as an extension is a subsignature and creates only a set constant along with a constraint making it a subset of each supersignature listed in the extension clause. All fields defined for a supersignature are automatically defined for the subsignature.

Sets & Set Operators

All signatures and relations in Alloy can be viewed as sets. A set contains several tuples of the same arity. There are three constant sets: **none** is the empty unary set, *univ* is the universal unary set containing all other sets, while *iden* is the set of all pairs relating every atom to itself.

```
21 -- *******Set Operators *******
22 + // union
23 & // intersection
24 - // difference
25 in // subset
26 = / equality
```

We concisely define the 5 listed set operators. A tuple is in a+b if and only if (iff) it is in a or in b. A tuple is in a&b iff it is in both a and in b. A tuple is in a-b iff it is in a but not in b. A set a is in set b iff ever tuple in a is in b too. a = b is true when a and b have the same tuples.

Relational Join

The most important relational operator in Alloy is called the relational join operator. In Alloy two tuples can be composed or joined if the last atom of the first tuple matches the first atom of the second tuple. The resulting tuple consists of the atoms of the first and second tuple, leaving the matching atom out. When two relations a and b are joined in a.b, the resulting relation is obtained by taking every combination of a tuple in a and tuple in b and calculating their relational join. We list the main other relational operators in Alloy:

```
27 → // The product of two relations
28 ~ // The transpose of a relation
29 ^ // The transitive closure of a relation
30 * // The reflexive transitive closure of a relation
```

Relational Product

The arrow product of two relations a and b is denoted $a \rightarrow b$, and is the relation defined as containing every possible concatenation of a tuple from a and a tuple from b. Consequently, when a and b are scalars, then $a \rightarrow b$ is a pair, when a and b are sets, then $a \rightarrow b$ is a binary relation.

Transpose & Transitive (Reflexive) Closure

The transpose of a relation r denoted by ~r is its mirror image, obtained by reversing the order of atoms in each tuple of r. Therefore a relation r is symmetric if and only if r= ~r. The transitive closure of a relation r is denoted $\hat{}r$ and is the smallest relation that is transitive and contains r.$\hat{}r$ is computed by the formula: $\hat{}r = r + r.r + r.r.r + ...$ A reflexive relation r is a relation that for every atom a it contains, also contains $a->a$. This can be expressed relationally as $iden$ *in* r (i.e., the identity relation is contained in r). Therefore the reflexive transitive closure can be expressed relationally by adding the identity relation to the transitive closure $*r$ as $*r = \hat{}r + iden$.

2.3.2.2 Standard Logical Operators and Quantifiers

Logical Operators

Logical operators are used in boolean expressions, and support two notations, a verbose (textual, on the left) and a short one (on the right).

```
31 not   ! // negation
32 and   && // Conjunction
33 or    || // disjunction
34 implies  => // implication
35 iff   <=> // bi-implication
```

Constraint Quantifiers

Alloy supports several first order quantifiers we explain underneath. Let x be a variable, e be an expression bounding x, and F be a constraint.

```
36 all x:e | F  // F holds for every x in e
37 some x:e | F // F holds for some x in e
38 no x:e | F  // F holds for no x in e
39 lone x:e | F // F holds for at most one x in e
40 one x:e | F // F holds for exactly one x in e
```

2.3.2.3 Facts, Predicates and Functions

Facts

There are various ways to express constraints on an Alloy model. Facts express constraints that must always hold on a model. Some facts are expressed in standalone fashion and some are directly placed into a special field of a signature which we call signature facts. The following fact says that all children must have exactly one father in two different ways. The first is expressed in the third signature *Child* as a signature fact, the second alternative definition is given as a separate fact at the end of the code snippet.

```
41 sig Person{}
42
43 sig Father extends Person{
44   children: some Person
45 }
46
47 sig Child extends Person{}{
48   some f:Father | this in f.children
49 }
50
51 fact{
52   all c:Child | one f:Father | c in f.children
53 }
```

Predicates & Functions

A predicate is a named constraint (returns a boolean), with one or many arguments. A function is a named expression returning a result, with one or many arguments. We give here an example of the definition of a predicate and a function. Predicates and constrains can be used to express other predicates and constraints, but also defined as assertions that must be checked on the Alloy models.

```
54 abstract sig Color{}
55 one sig Red, Green extends Color{}
56 sig Light{}
57 sig LightState{
58   color: Light →one Color
59 }
60
61 pred lightIsRed[s:LightState, l:Light]{
62   l.color.s = Red
63 }
64
65 fun redLights[s:LightState] : set Light{
66   s.color.Red
67 }
```

We define two *colors* (*Red* and *Green*), and a signature *LightState* which contains a relation *color* mapping *Lights* to one *Color* each. The predicate *lightIsRed* holds if and only if the *Light* l is *Red* by using a relational join. The function *redLights* returns a set of *Lights*, which are currently *Red*. It computes this set as the result of a left outer relational join of the *LightState* with the *color* relation then with the *Red* unary signature.

2.3.3 The Alloy Analyzer

Alloy [Jac12] is presented as a so-called Lightweight Formal Method [JW96]. The analysis of Alloy model is a form of constraint solving. First, there is *Simulation*, which involves finding instances of executions (e.g., states) that satisfy a given property. On the other hand, *Checking* involves finding a counter-example, i.e., a model instance which breaks a given property modelled as an assertion.

Both types of analysis with Alloy are conducted within a state whose size is specified by the user in a *scope*. A *scope* bounds the number of atoms of each signature which can be instantiated. The scope by default is always 3. One of the key techniques in using the Alloy analyzer well is to find a minimal scope required for each signature in order to be able to instantiate satisfiable models while finding property violations.

Alloy works under the small scope hypothesis, which assumes that most bugs or errors in a program may be found using very small counter-examples. The state space bounding makes the usage of Alloy 'lighter' than exhaustive approaches such as 'pure' model checking, and the tool is quicker to construct these (smaller) state spaces and find possible violations. However, due to this state space bounding, not finding an error in the Alloy model does not imply that the model is correct, as an error may be found if the state space is grown bigger to take more cases into account. This is why most of the time, it is reasonable to assume that the higher the scope, the more confidence we can have in the result of the checking of the Alloy Analyzer. Let us run the previous example with *Lights* and *LightStates* and visualize what the Alloy Analyzer generates.

```
68 pred show{}
69 run show for exactly 2 Light, exactly 2 Color, exactly 4 LightState
```

We define the special predicate *show*, then run this predicate while giving a scope as a parameter. In this alternative, we specify the exact number of instances of each signature we would like to generate. In the third line of the code snippet, we show a simpler way to set a scope to all signatures at once. The obtained result is shown in Figure 2.4. We can see four different *LightStates*, two *Lights* and two *Colors*. Each *LightState* possesses two instances of the *color* relation from *Light* to a *Color* (i.e., *Green* or *Red*).

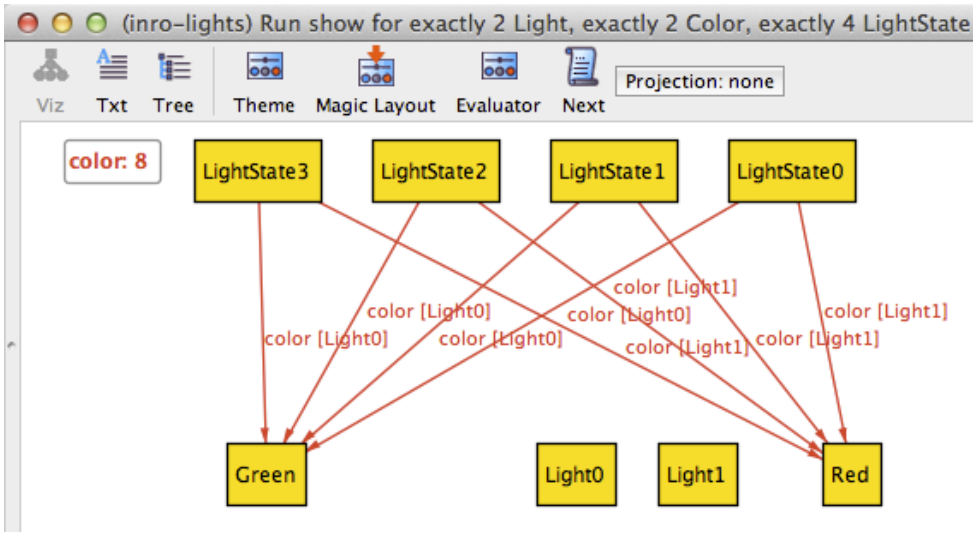


Figure 2.4: Alloy Analyzer Instance for The LightState Model

Now let us model a trivial assertion we know is invalid. Let us assert that all *Lights* are always *Red* in all *LightStates*.

```
70 assert lightsAreAlwaysAllRed{
71   all l:Light | all ls:LightState | lightIsRed[ls,l]
72 }
73
74 check lightsAreAlwaysAllRed for 4 but 2 Light
```

We then run the Alloy Analyzer to check this assertion for a maximum of 4 instances per signature, but 2 instances of *Light*. Unsurprisingly, we find a counter-example. In Figure 2.5 a state is visualized, where the only *Light* in the instance has the color *Green* in the state *LightState0*.

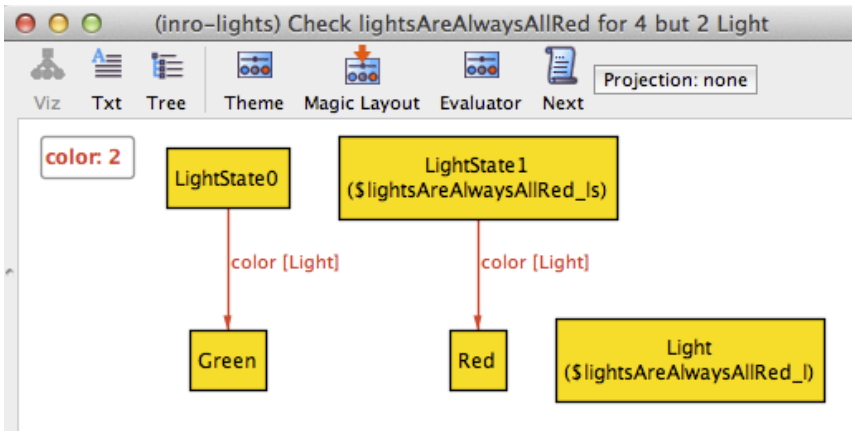


Figure 2.5: Alloy Analyzer Counter Example to the lightsAreAlwaysAllRed Assertion

2.3.4 Comparison of Alloy with Other Formal Methods

The semantics of Alloy is given in Boolean Formulas. Alloy transforms a model into a conjunctive normal form formula and so building the state space defined by an Alloy model reduces to a SAT problem. Alloy supports several alternative SAT solvers such as SAT4J [UC13], MiniSAT [ES13] and ZChaff [aP13] which check the satisfiability of propositional logic formulas.

The Alloy analyzer cannot be said to be a classical model checker, and even less a theorem prover [Jac02]. Model checkers [CK08] such as SPIN [GJH03] or NuSMV [CCG+02] build a finite but comprehensive state automaton or labelled transition system representation of all possible executions of the system. They take additionally as an input a temporal formula and use various means to check the acceptance of this formula by the formal representation of possible system executions. Model checkers classically suffer from the state explosion problem, but make use of various optimizations to produce answers more quickly.

On the other hand, theorem provers such as CoQ [INR13] or PVS [PVS13, BK06] are interactive tools for deduction-based automated reasoning [HR00]. When failing to prove a theorem it is often difficult to see whether the theorem is invalid, or whether the proof strategy failed. The augmentation of a theorem proving approach with Alloy has been considered in [MB02] to mitigate this.

Depending on the specific domain, the properties that can be analysed using Alloy classically range from reachability and absence of deadlocks, e.g. in a ring protocol, to application-specific soundness of systems, e.g. someone in the enterprise is always able to have access to the critical databases of the company [Jac12].

Chapter 3

State of the Art in Regulatory Compliance Management

Man cannot remake himself without suffering, for he is both the marble and the sculptor.
Dr. Alexis Carrel.

3.1 Introduction

Compliance management deals with ensuring that a given enterprise is in accordance with a set of regulatory guidelines. Being 'in accordance' refers to a specific state or an interval between two states in the history of the evolution of enterprise. Regulatory compliance management (RCM) deals with the modeling, checking, enforcement, and analysis of compliance requirements (CRs) extracted from regulations of various kinds, such as laws (i.e., legislations), contracts, internal policies, etc. The RCM community takes influences from various areas such as information systems, software engineering, artificial intelligence, requirements engineering and law.

Since the year 2000, there has been a surge in interest in RCM from research and practice. This is due, at least in part, to the new wave of regulations such as Sarbanes-Oxley[1] and Basel II[2]. These laws were defined and enacted in response to the series of financial scandals that shook the economic sphere[3] and signified a change of philosophy from total self-regulation under the free market paradigm towards different degrees of regulatory control.

As Abdullah et al. [Abd10] put it: "failing to comply is no longer an option [AHK07, BMG08]". In [JB06b], AMR Research mentions three kinds of compliance companies try to address: regulatory, commercial and organizational. In the definition of RCM we give in chapter 1, regulatory compliance is defined as subsuming all other kinds of compliance (e.g., legislative, contractual). Albeit regulatory compliance in [JB06b] actually means legislative (i.e., to law) compliance, what is important is that already in the mid 2000s, the RCM market was expected to grow as a strategic enterprise concern and provide growth opportunities, reduce risk and enable anticipating new compliance requirements.

Works on RCM have been surveyed and analyzed from various angles. In [Con09] and [ea09a], a detailed domain analysis for regulatory compliance is given, which is narrower

[1] www.soxlaw.com european resp. japanese 'variants' are also known as EURO-SOx and J-SOx
[2] http://www.bis.org/publ/bcbsca.htm
[3] Non-exhaustively referred to are often: Enron (USA), Parmalat (Italy), Verizon (USA), Societe Generale (France), HIH (Australia), etc.

than in our sense (cf. Section 3.3) but clearly shows the disparity in views on RCM as well as the great variety in regulations and CRs. Abdullah et al. [ASI10, Abd10, AIS09] and Cleven et al. [CW09] proceeded to a systematic literature analysis of existing works from the perspective of information systems. Ly et al. [LGRMD08, LRMGD09] and El Kharbili et al. [EKAdMSvdA08] give an overview and analysis of RCM from the perspective of BPM. Turki et al. [TBO10] give a short summary of research around RCM from the perspective of service computing while Otto et al. [OA07] survey over 50 years of efforts in handling legal texts for systems development from a requirement engineering perspective.

In the context of BPM, full coverage of RCM means full coverage of the BPM lifecycle [EKAdMSvdA08, LRMGD09]. Cleven et al. [CW09] bring additional light in this matter by proposing a business engineering analysis framework that uses a layered view on RCM distinguishing the strategy, organizational/process, IT/Business alignment and IT implementation layers. This is also supported in [EKSMP08]. Related to this is the thorough review of existing candidates for a compliance modeling language in [Uni08] and [CFPC09], to which a separate contribution made in [ETvdHP10a] and [OA07] can be added. This work has been preceded by an extensive overview of candidates for policy languages which could be used for modeling compliance using business rules in [PAN04].

In the remainder of this chapter, we proceed as follows. Section 3.2 provides the setting for a common understanding of the RCM domain by stating definitions for the most important domain concepts. This is followed by the proposal of a conceptual framework for automated RCM in the context of BPM in Section 3.3. Section 3.4 explains the strategy followed in the survey of approaches to RCM for BPM we realized. Section 3.5 gives a concise description of the works in the survey. Section 3.6 shows the result of our evaluation and comparison of the surveyed approaches. Finally, Section 3.7 discusses our findings and suggests directions for future research around RCM related to the surveyed solutions. Section 3.8 concludes this chapter.

3.2 Background Definitions

First, in an attempt for us to strictly delimit the relevant RCM concepts and their intended semantics, we define what a regulation is and, building upon intermediary definitions, define RCM for BPM. We built the definitions given underneath from the existing definitions in both research and practice, in an attempt to make the definitions as generic and implementation-independent as possible. We find similar and complementary definitions in [EKSMP08, LRMGD09]. This is not surprising since these works tackle RCM in the context of BPM. The definition of RCM given by AMR Research[4] [JB06b] is the closest to ours.

Definition 3.2.1 (Regulation)**.** *A regulation is a document written in natural language containing a set of guidelines specifying constraints and preferences pertaining to the desired structure and behavior of an enterprise. A regulation specifies the domain elements it applies to. Examples of regulations are a law (e.g., the health care law HIPAA[5]), a standardization document, a contract, etc.*

Definition 3.2.2 (Regulatory Guideline)**.** *A regulation guideline specifies the expected behavior and structure on enterprise domain elements. It additionally defines tolerated and non-tolerated*

[4]Now known as Gartner Research after acquisition by the latter.
[5]Health Insurance Portability and Accountability Act. http://www.cms.gov/HIPAAGenInfo/

deviations from the ideal behavior and structure, and also defines the possible exceptional cases. A regulatory guideline may also specify how the enterprise ought to or may react to deviations from ideal behavior and structure. Note that in the law domain, regulatory guidelines are referred to as norms.

The following example 3.2.1 shows a privacy regulation from the healthcare domain. It is structured so that elementary regulatory guidelines can be separately understood by the regulation reader. This regulation describes how *covered entities* must behave regarding disclosure of health information about individuals.

Example 3.2.1 (HIPAA Regulation)**.**
Excerpt [fMS13, TMA06] from §164.520(c)(2) and (c)(3)]

(a)Standard: *The covered entity must provide the individual notice.*

(a)(1) *A covered entity who has a direct treatment relationship with an individual must:*

(a)(1)(A) *Provide notice no later than the first service delivery;*

(a)(1)(B) *If the covered entity maintains a physical delivery site:*

i. *Have the notice available for individuals to take.*

ii. *Post the notice in a clear and prominent location.*

Regulations are usually decomposed in compliance requirements (CRs), which are text snippets containing separate guidelines. Therefore CRs are the material (textual) occurrences of the regulatory guidelines that the regulation expresses. Basically, the two notions are equivalent, regulatory guidelines are the intention or desire of the regulator, and CRs are the understanding of the compliance manager. CRs are the work of a compliance manager, who extracts and organizes them as he wishes from the regulation. The granularity of such CRs is left to the appreciation of the regulation expert who extracts them.

Definition 3.2.3 (Compliance Requirement (CR))**.** *A compliance requirement (CR) is a piece of text extracted from a regulation that specifies a given regulatory guideline. It may refer to or be related to (e.g., through exception relations) other CRs.*

CRs must be interpreted by regulation and business experts in order to be transformed into a form that makes them understandable to and enforceable on the enterprise. This process is called concretization. It is also sometimes called contextualization or internalization of CRs in the RCM literature.

Definition 3.2.4 (Concretized CR)**.**
A CR is interpreted and expressed in a form that allows relating it explicitly to an enterprise model (e.g., business process model).

These concretized CRs can be refined together by regulation experts and business analysts until they reach a state where they explicitly refer to actions and elements defined in the enterprise model. In the rest of the thesis, we will asbtract from this distinction and refer to both CRs and concretized CRs as CRs.

Example 3.2.2 (Concretized CR)**.** *For the following compliance requirement:*
Medical institutions are not allowed to conduct treatments or examinations on patients if these constitute any danger to the health or unnecessary convenience to the patient.

We give the corresponding concretized CR:

In Hospital X, in Service Y, no endoscopic examinations shall take place within one week after radiological examinations using non-water soluble contrast agents. If that happens, a chief physician has to look at the case.

If we take into account the previous definitions, we can restate the high-level RCM definition 1.2.1 by restating it into a working definition in the context of an enterprise for the rest of this thesis:

Definition 3.2.5 (Working Definition of the RCM Problem)**.**
Regulatory Compliance Management (RCM) is the problem of ensuring that enterprises (data, processes, organization, etc.) are structured and behave in accordance with the regulations that apply, i.e., with the compliance requirements specified in the regulations. In the opposite case we say that a company is violating a regulation. RCM is composed of compliance modeling, checking, analysis and enactment.

We divide the RCM problem (see definition 3.2.6) into two sub-problems.

Definition 3.2.6 (Two RCM sub-problems)**.**
1. ***Compliance Modeling:*** *The problem of (accurate) representation of CRs.*
2. ***Compliance Checking:*** *The problem of statically (verification) or dynamically (enforcement) verifying whether or not a given enterprise model fulfills the CRs and/or the problem.* □

3.3 Conceptual Framework for Business Process RCM

RCM involves several tasks, which we elicit according to the conceptual framework given in this chapter (cf. Figure 3.1). In this framework, an enterprise model describes the enterprise, while the behavioral part of the enterprise is represented by its business processes (BPs) as the means describing the behavior of an enterprise. This framework seeks to provide a foundation for automated RCM starting from RCM modeling, and extending this foundation to fulfill the objectives of ensuring and maintaining control over compliance.

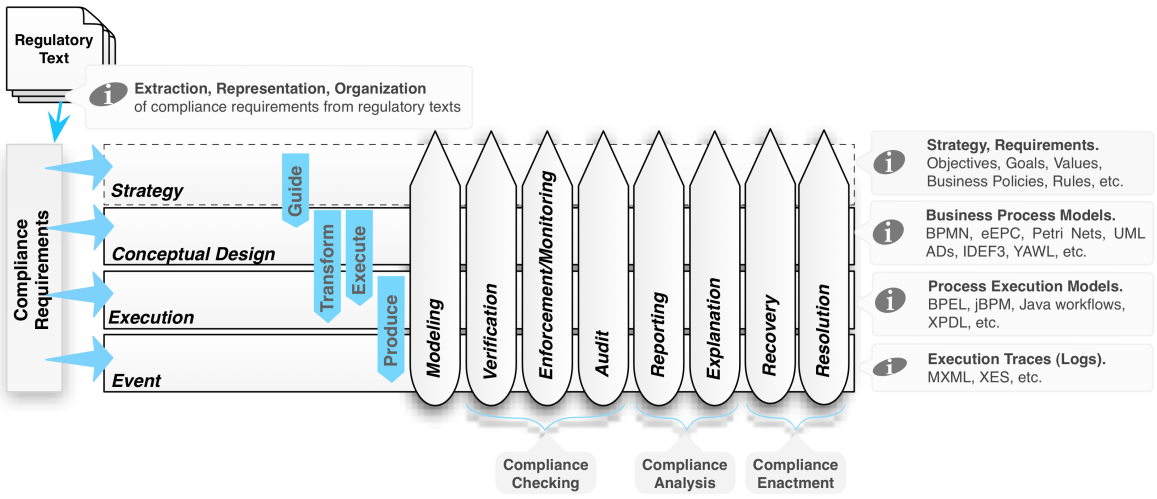


Figure 3.1: Automated Regulatory Compliance Management - Conceptual Framework

3.3.1 The Business Process Management Dimension

In Figure 3.1 the conceptual framework used in this chapter for evaluating RCM approaches is presented. We introduce here the artifacts present in each of the four horizontal BPM layers from 3.1: (i) Strategy, (ii) Conceptual Design, (iii) Execution and (iv) Event.

Definition 3.3.1 (Strategy Model in the Strategy Layer)**.**
The strategy models describe the intentions and motivations behind the definition of the business process models. It includes concepts such as business objectives and goals, values and value chains, risks, business policies and business rules, strategic actors and dependencies between the latter. There exists a variety of languages following different modeling approaches at the strategy level (e.g., [FMPT01]).

Definition 3.3.2 (Business Process Model in the Conceptual Design Layer)**.**
A business process (BP) is a collection of tasks that together reach a business goal, such as creating value, producing a service or a product. A BP model is a representation of a BP that describes all possible executions as an orchestration (i.e., ordering as a workflow) of BP tasks. In our definition, a BP model also describes the involved organizational entities and their roles in achieving business goals, as well as the used data and resources. Alternative definitions are provided in [Wes07, AH02, Sch99, OMG11].

Definition 3.3.3 (BP Execution Model in the Execution Layer)**.**
Some BP modeling language allow creating BP models which are executable, such as BPEL or XPDL. A BP execution engine is the software responsible for executing BP instances. Every BP model has several (possibly concurrently) executing instances.

Definition 3.3.4 (BP (Execution) Trace in the Event Layer)**.**
While being executed, a BP model instance generates a sequence of event logs describing the events which occurred during execution, such as finishing a BP task or an exception that occurred. The complete sequence of event logs (i.e., after execution termination) is called an execution trace. Software exists for extracting these event logs and representing them graphically in notations similar to that used for BP models [RvdA06].

3.3.2 The Compliance Dimension

We give in the following definitions a high-level introduction to the elements of the conceptual framework in the vertical dimension, as represented by the eight vertical arrows in Figure 3.1.

3.3.2.1 Compliance Modeling

The foremost task in RCM is the formal representation of compliance requirements in a form that makes them machine-interpretable. In this task, the input is a structured and refined representation of compliance requirements extracted from the regulation. Usually, this extraction requires the intervention of (i) regulatory (e.g. juridical) and (ii) enterprise experts (e.g. business analysts).

The sound legal understanding of statements (e.g., liabilities, responsibilities, duties, authorities, etc.) contained in the regulation and their correct normative interpretation require skills that pertain to the domain of a regulation. This is called *Legal Interpretation*. Take for example a regulation that is a law (e.g., HIPAA respectively SOx), it requires the intervention of a law (e.g., healthcare law respectively financial law) expert.

Additionally, the *Enterprise Context* also plays a role, as the intervention of an individual with profound knowledge of the enterprise that is impacted by the regulation is required in order to relate the regulation to the elements of the enterprise model that are covered by the CRs. Consequently, the same regulation can, in the context of two different enterprises, or by two different legal experts, lead to two different interpretations thus yielding two different compliance models. This procedure is called *concretization* of the CRs (cf. definition 3.2.4).

The produced CRs are in the form of text snippets. They must be related to each other in order to determine how the interpretation, respectively the enforcement of the one CR is dependent on the interpretation, respectively the enforcement of the other regulation CRs. Once this structured representation is available it must be transposed into some formal representation, for which there is a wide variety of languages (e.g., logics, XML-based, etc.) available [OA07]. Moreover, the CRs must be interpreted in the relevant *Enterprise Context*. The obtained structured, and interpreted representations of CRs are called *Concretized Compliance requirements*.

Definition 3.3.5 (Compliance Modeling)**.**
The task of (accurate) formal representation of CRs (if necessary, concretized) in the context of a given enterprise (i.e., for a given enterprise model).

Another element of compliance modeling is compliance validation.

Definition 3.3.6 (Compliance (Requirement) Validation)**.**
Validating the consistency of the formalized compliance requirements.

In definition 3.3.6, by consistency we mean more precisely: non-redundancy and conflict-freedom. In some approaches, mechanisms may be defined to ensure conflict freedom through conflict resolution while reasoning on the CR models. The latter approach is dynamic, while in some approaches, logics allow to find out statically whether a set of CR models is consistent.

Compliance validation seeks to help the user answer the question of whether the CRs are coherently and consistently modeled. But it does not answer the question of whether the CRs are adequately modeled, i.e., the CRs were modelled as they were meant to be.

A related task is CR normalization in which a set of CR models is reduced to a new minimal set of CRs which is equivalent to the first set. But this is usually not a task that a compliance modeler should do, and is mainly motivated by the search for smaller CR models which can be more efficiently (quickly) verified. Usually, this task is done at the rule level. In some approaches normalisation is more than reducing the set of CR the minimal one. For instance, in PCL [GR10b] the idea behind normalisation is to make explicit all formally implicit requirements and then remove redundancy. One of the reasons for this is proper and accurate modelling. This allows the proof system to avoid reporting non-compliance in case of a violation of a rule/norm that can be compensated.

3.3.2.2 Compliance Checking

With the formally modeled CRs at hand, it is possible using a variety of technologies, which depend on the formalism of choice for representing the CRs, to conduct automatic checking of compliance of the enterprise model (e.g., a BP model) to a set of CRs. Compliance checking can be of four types shown in Figure 3.1: compliance verification, enforcement, monitoring and audit.

Definition 3.3.7 (Compliance Checking (C^2))**.**
The task of deciding *whether or not a given enterprise model* fulfills the compliance requirements. *It is the process of conducting either one or both of compliance verification (denoted C^2_S) and compliance enforcement (denoted C^2_D) defined later in this document. This step may include providing a* ***proof of compliance****, and most of the time, a* ***proof of violation*** *(either a counter-example or a logical argumentation demonstrating that compliance cannot be guaranteed) is highly desired. It may also include a* ***localization*** *and/or and* ***explanation*** *for the uncovered violation.*

The problem of *compliance checking* can be modeled as the problem of deciding on the truth of: $EM \vdash CRs$. This formula states that the enterprise model (EM) satisfies a set of CRs. This 'binary' view on compliance is an absolutist definition of compliance. Some approaches to compliance define quantifications of compliance as compliance degrees [GR10a] which allows to consider several possible states of compliance. The BP model must be represented in a formalism on which statements of the formalism in which CRs are modeled can be interpreted. The algorithms to decide on the value of $EM \vdash CRs$ vary depending on the compliance modeling formalism. For example, if CRs are represented as temporal formulas written in CTL [CK08], then the EM must be transformed into a formal model for interpreting the logical CTL formulas (e.g., *Kripke Model*) and a model checking algorithm must be used.

In an extended fashion, compliance checking can be seen in its core as a partial function C^2_{CRs} that for a given set of CRs, maps an EM and the state the EM finds itself in, to a set of violations. Be **EM** the set of EMs and be **CR** the set of all CRs: $\mathbf{EM} \times StateSpace_{EM} \times 2^{\mathbf{CR}} \mapsto 2^{ViolationSet}$. The *ViolationSet* may be a singleton. This set may be empty, in which case the EM is said to be fully compliant and against a given set of CRs.

Compliance Verification checks whether an EM (e.g., BP) will always be compliant with the CRs no matter which instance we look at. It is typically a costly process that requires powerful formal methods such as theorem proving or model checking.

Definition 3.3.8 (Compliance Verification (C^2_S))**.** *The problem of* statically verifying *whether or not a given business process model fulfills the CRs. It is classified as a pro-active compliance checking technique [EKSMP08].*

Compliance Enforcement checks whether the current EM instance being executed is about to violate the CRs and to react accordingly by using enforcement mechanisms that will modify the behavior of the instance in order to avoid the violation (e.g., not authorizing access to a given resource by a BP task). In the software security community, many works seeking to embed access control in workflows have been realized (e.g., [BMPS10, WM10]).

Definition 3.3.9 (Compliance Enforcement (C^2_D))**.** *The problem of* dynamically enforcing *CRs on a running business process instance. This step may include providing a proof of compliance. It may also include a localization and/or and explanation for the uncovered violation. It is classified as a pro-active compliance checking technique.*

Compliance Monitoring is the passive variant of compliance enforcement and uses mechanisms to observe the current incomplete execution history of the BP model in order to detect violations that have already occurred. As such, it is different from compliance enforcement in that it is a detective task and not a preventive (check meaning of pre-emptive) task. An Example of a formalism used for realizing compliance monitoring is event calculus (interpreted over an event trace produced by a BP execution).

Definition 3.3.10 (Compliance Monitoring)**.** *The problem of continuously observing the history of execution of a business process instance while it is still running and deciding whether or not a compliance violation happened. It is classified as a passive compliance checking technique.*

Compliance Audit is the compliance checking of a single (or many) complete (finished) process execution trace. As such it can be seen as a simplified version of compliance verification where the BP model to be verified is a single sequence. Linear-time temporal logics (LTL) or event calculus can be used for this task. It can be very useful when simply auditing enterprise models and taking samples of EM elements, i.e., terminated BP model instances. Its conclusions about compliance is however less meaningful that for compliance verification.

Definition 3.3.11 (Compliance Audit)**.** *The problem of observing the total execution history (trace, log) of a business process instance that has terminated and deciding on whether or not compliance violations occurred. It is classified as a passive compliance checking technique.*

3.3.2.3 Compliance Analysis

Compliance Analysis consists of the use of compliance reporting and/or traceability as defined underneath.

Definition 3.3.12 (Compliance Reporting)**.** *The task of generating documentation, traces and (visual) analytics related to the results of compliance checking.*

Definition 3.3.13 (Violation Traceability)**.** *We define Violation Traceability as Explanation, respectively Localization of violations. It seeks to provide the causes (events) that led to the occurrence of the violation, respectively the elements of the BP model where the violation occurred.*

3.3.2.4 Compliance Enactment

Compliance Enactment consists of the use of compliance recovery and/or resolution.

Definition 3.3.14 (Compliance Recovery)**.**
The problem of defining and executing automated mechanisms to dynamically (at run-time) react to the occurrence of a violation, and re-establish a state of compliance through either a handling, compensation or reparation action. It is used together with compliance enforcement or monitoring.

Violation Recovery mechanisms are useful at both design- and run-time in assisting users in resolving violations using pre-defined recovery specifications, sometimes available from best practices. *Violation Recovery* can be extremely useful during enforcement to assist in automating compliance management without requiring the intervention of humans and resume execution of business processes. An even higher level of RCM maturity in an enterprise is reached if violation resolution is supported.

Definition 3.3.15 (Violation Resolution)**.** *It is the task of engaging corrective action to remove the cause(s) of the occurred violation(s) in order to re-establish compliance. It is not automatic as in* violation recovery *and is engaged upon failed compliance verification or audit.*

We propose to deal with the RCM problem by following a model-driven and policy based approach. This will allow us to build on existing approaches and languages in order to combine CR modeling, verification at design-time and enforcement at run-time.

3.4 Analysis Design

Most existing surveys of RCM approaches for BPM do not contribute a comparative analysis of the solutions but instead, proceed to an initial study of available research in order to structure approaches but do not extract a detailed set of functional and non-functional requirements on RCM for BPM. Our work is complementary to existing RCM surveys, as we specifically focus on the *capabilities* of existing *solution* papers to RCM as well as how they fit into the *RCM conceptual framework* given in Section 3.3.

Our survey was realized in order to help achieving a broad understanding of RCM and provide researchers with various perspectives on RCM with a common ground for discussion. This can be done by providing a chart of existing works and showing how they tackle RCM, as well as uncovering the respective limitations. This paper answers the need demonstrated in [Abd10] for an "elementary study of the components and 'state of the art' features that should be incorporated in compliance management tools, i.e., frameworks, guidance and software".

We abstained from conducting a systematic literature review as in [AIS09, CW09] by, for example, selecting the most visible journals in the area of information systems (IS) and BPM. Instead, we used a next-to-next strategy starting from related research to ours to find and select papers that are relevant for our analysis. The criterion for selection is that the paper explicitly mentions tackling the RCM problem for BPM and proposes a solution for it. Unsurprisingly, the wide majority of the papers we retained have been published at workshops, conferences and in PhD theses. We collected a substantial number of approaches, each published in one or many papers, of which we include 32 approaches in this chapter.
Unlike systematic literature surveys where selected venues and search keywords are specified before-hand, our approach requires prior knowledge about the papers that are selected. As such, our selection strategy may not be exhaustive but rather, is very focused on RCM solutions in the domain of BPM. Also, we did not purposefully consider approaches solving the problem of *compliance requirement extraction and elicitation* from regulations as we concentrate on solutions for the problem of *modeling and checking compliance, as well as violation recovery.*

We study the perspectives taken on RCM by researchers and build a common taxonomy of evaluation criteria that RCM solutions attempt to fulfill. Eventually, such a taxonomy of requirements is to be validated and extended by the community in order to be used as an initial set of criteria against which to evaluate existing RCM solutions and that can be used for discussion among researchers on RCM and be possibly extended. Practitioners and researchers attempting to develop RCM solutions that are generic, flexible and holistic can hence benchmark the existing solutions against the relevant subset of criteria that is of interest to their concrete RCM concerns.

3.5 Overview of Selected Publications

Before showing the result of comparing the various surveyed approaches, we begin by giving a short summary of the works that have been considered in the comparative analysis contributed in this thesis. We assign an ID to every RCM solution which will be used in the comparative analysis tables below to refer to it. A very short description of every work stream is given underneath.

[1] Awad et al.[Awa10, ASW09, AWW11, AWW09, ADW08, AGTW11] provides a comprehensive work on regulatory compliance for BPs in his PhD thesis [Awa10] and related publications. His approach tackles rule modeling, rule consistency analysis, rule verification, violation localization and heuristics for violation resolution. The rules are modeled using a language called BPMN-Q that allows to graphically draw queries to be executed against a BP model repository. BPMN-Q rules have formal semantics in LTL and CTL. BPMN-Q supports combining temporal dependency constraints and data constraints to form a particular blend allowing to express compliance rules on the behavior and structure of an enterprise.

[2] Liu et al. [LMX07] propose a method for BPEL process verification against graphically modeled rules in a language called BPSL. This language has very interesting properties such as default compliance conditions, logical and business templates. BPSL rules (called properties) also support data conditions. The BPEL processes are then transformed into Pi-Calculus for analysis and further into FSMs for verification against LTL temporal formulae generated from the BPSL rules.

[3] Ly et al. [LRMGD09, Ly13, LGRMD08] use event traces left resulting from the execution of business processes and carrying context information to provide a logical semantic model for the definition of compliance. However, the constraints taken into consideration by this work only include simple task dependencies, such as mutual exclusion or presence dependency. The particularity of Ly et al.'s work is that they study in particular adaptive processes, i.e., processes which can can be changed at the model (process template) or at the instance level. This leads to a new set of problems related to compliance regarding the propagation and consistency of changes.

[4] Governatori et al. [GMS06, LSG08, SG10, GR10a, LSG07, Gov05, KGS10] rely on the principle of compliance by design, which consists of ensuring that BP models are compliant before these are deployed and executed. Governatori et al. use formal logic to provide a logic inference-based solution to compliance checking. They rely on the definition of internal controls using the formal contract language. This language has a native operator called the contrary to duty operator that allows to define violations and chains of violations. Thus, a logical mechanism for violation resolution is provided. This approach relies on manual annotation of process activities with effects and thus checking compliance is reduced to the propagation of these effects through all possible execution paths. A process model is thus compliant if none of the possible executions leads to effects which are violating the compliance requirements modeled as FCL rules.

[5] El Kharbili [EKSMP08, EKP09, ES09] propose the use of business rules written in the WSML-Flight semantic web rule language for checking business rules on BP models created

using ontologies for BPM. The authors use the decision point metaphor, where BP models are semantically aligned to compliance requirements by attaching business rules to decision points in a BP model.

[6] Weber et al. [Web09, HWG09] use an approach similar to Governatori et al., in that process tasks are annotated with effects, compliance requirements modeled as conjunctions of propositions and algorithms designed to check compliance in polynomial time are defined. However, this approach does not allow full normative reasoning with loops (cycles) even though the work version in [HWG09] includes heuristics for dealing with loops.

[7] Namiri et al. [Nam08, NS07b, NS07a, SGN07, NS07c] uses a similar approach to El Kharbili, by following the decision point metaphor, and modeling compliance requirements as business rules in SWRL, a semantic web rule language. Data about the BP model is extracted and transformed into facts and stored into a knowledge base external to the BP execution engine. However, it is not clear how users can create business rules out of regulations using this approach, since the SWRL rules here are technical. One interesting feature in Namiri's approach is that he defines compliance patterns.

[8] Schmidt et al. [RCR07] also propose the use of ontologies to represent enterprise knowledge, and use logical statements to express constraints. Using ontology-based reasoning, the authors exploit the fact that if it is possible to create instances of the ontology thus defined then the model is compliant.

[9] Elgammal [ETvdHP10b, TEvdHP12, OAvdHWM11, ETvdHP10b, Ama12] defines temporal dependency patterns between process tasks extending the well-known Dwyer property specification patterns [MGJ98] and gives them semantics in LTL. She then uses root-cause trees to link the possible violation of a temporal pattern to a possible cause. She thus seeks to assist business users (BUs) in getting explanations to compliance. However, root-cause analysis requires the user to manually plan and create these trees. As such, these trees are application-dependent and must be redefined for each new application needing compliance checking. The trivial part in root-case analysis is the association of a temporal pattern evaluated to False to a set of possible terms in the pattern instance that were evaluated to False. Thus, trivial logical reasoning can somewhat help the BU in narrowing down the possible causes for a violation.

[10] Kokash et Arbab [KKdV10] use the REO coordination languages to define a formal representation of BP models. REO is a channel based language for defining logical circuits, and can be used to define execution and, using constraint-base automata, define data operations. Using REO and the mCRL2 model checker, BPMN models are verified using model checking.

[11] Goedertier & Vanthienen [GV06] introduce PENELOPE, a language for expressing timing and temporal dependency rules about the obligations and permissions in a business interaction. They use PENELOPE to declaratively capture the compliance requirements in regulations and provide an algorithm to generate BPMN process models from PENELOPE specifications. The use of such a technique is limited to the stage where business users (BUs) do early requirements elicitation and the generated BP models are not intended for productive use.

[12] Ghose et Koliadis [GK07] abstract from any compliance requirement modeling language and provide heuristics to resolve compliance violations using the notion of compliance patterns, thus yielding a semi-automatic way for compliance resolution. The heuristics introduced are

defined based solely on structural aspects of BP models and are not leveraged to the business level.

[13] The REALM approach by Giblin et al. [GLM+05] introduces a real time object temporal logic which allows very expressive compliance requirements to be modeled. But no implementation of an engine for the introduced logic is given. However, the whole approach behind work on REALM distinguishes itself from other works by focusing on various enterprise business aspects, and targeting genericity by generating compliance checking artifacts from the compliance requirements specified in the real-time object temporal logic.

[14] Pesic et al. DecSerFlow [Pes08, MPvdAP08, PSSvdA07] proposes an approach based on a declarative graphical language with semantics in LTL to model service behavior, and generate service orchestrations that act as business processes. Such BP models are compliant by construction.

[15] Yip et al. [FPR07] belong to the group of approaches based on ontologies, and use an OWL ontology to model the system to be checked for compliance. Compliance requirements are represented using the SWRL semantic web rule language.

[16] Padmanabhan 2006 [PGS+05] is also a logic-based approach, like Governatori et al. defining a modal logic based on commitments while Governatori's FCL is based on deontic logic's obligations. Note that Goedertier & Vanthienen's PENELOPE is also based on commitments that agents hold towards each other and that must be fulfilled.

[17] Foerster et al. [FESS07, FES05, FESS06] uses quality patterns modeled using a stereotyped UML activity diagram notation. Process models in UML activity diagrams are translated into transition systems and the quality patterns that express compliance requirements are transformed into temporal properties. Compliance checking is done using model checking.

[18] Kumar et Lui [KL08] use role patterns in order to model a specific class of compliance requirements. The role patterns concern role operations and task allocation, and as such, they are particularly well-suited to organizational compliance requirements, e.g., role-based access control or segregation of duty. Semantics of role patterns are given in Prolog in the paper.

[19] Wolter et al. [WSM08] link their work on task-based entailment constraints to the well-studied workflow patterns. However, they only cover a subset of these patterns. They provide an elegant graphical notation that can be embedded in a business process modeling notation, as is illustrated for BPMN. It is shown that ambiguities arise due to the nature of complex control flow patterns.

[20] Agrawal et al, [RCJL06] propose using database technology for compliance management in workflows. They explain how available technology can be applied to various compliance management tasks, and illustrate their ideas using a real-life example for the Sarbanes-Oxley regulation.

[21] The COMPAS[6] research project [Uni08] is a European research project around model driven compliance for SOA and BPM and was started beginning of 2008, and finished in beginning of 2011. In Compas, a variety of technologies are used for compliance modeling,

[6] http://www.compas-ict.eu/

checking, and monitoring. The works of Schumm [DON+10], ElGammal [ETvdHP10b], and REO-based compliance [KKdV10] are all outputs of this project.

[22] Schaad et al. [SM02] tackle compliance from the organizational control perspective. That is, they care about such principles as separation of duty, supervision, review and delegation among roles. The semantics of an enterprise model are given in terms of states and state sequences in Alloy, and of organizational control compliance requirements in terms of Alloy predicates. Alloy is a model checker that uses SAT solvers and relies on the small scope hypothesis to uncover specification errors.

[23] In [SLS06], Schaad et al. follow a different approach to model and check compliance requirements, in the case of safety properties of a process. The authors present a model-checking approach which also supports delegation and revocation of tasks. The approach is implemented using the SMV model checker using the LTL formal language.

[24] Gahanavati [SAD+10] tackle RCM from a goal-oriented requirements engineering (GORE) perspective. Their definition of a regulation, generic, and unlike other GORE works, does not concentrate on security or privacy aspects. The objective here is to integrate law modeling notations (i*, a GORE notation) with business process modeling notations (Use Case Maps). Traceability and responsibility links are drawn between the regulation model and the process model. This approach claims helping assess compliance. We rather see it as a documentation and traceability analysis approach, which allows to provide a quantification of compliance once compliance to various parts of a regulation are already evaluated and quantified. This framework also allows to accompany change in regulation by enabling impact analysis on BP models.

[25] Desai et al. [DNS08, NAAS05] define and formalize contract correctness criteria based on the preferences of agents involved in a contract. Through a logical formalization of the notions of safe and beneficial contracts, partners in a contract can check the consistency of the contract. Algorithms for checking the safety and benefits of a contract for an agent are given.

[26] Kuester et al. 2007 [JMKH07] consider the (data) objects used as inputs and outputs of BP tasks. Tasks change the state of the input objects when outputting them. Objects have their own lifecycles already modeled. The question raised is whether BP models operate on the objects in conformance to the objects' lifecycle or not. A technique for answering this question is given, and an algorithm for generating a BP model from a set of object lifecycles is provided.

[27] Yu et al. [YMH+06] propose a (temporal) property pattern based specification language, named PROPOLS, whose semantics are given in Dwyer's property specification patterns, which in turn have formal semantics in LTL. PROPOLS is used to verify BPEL service compositions. PROPOLS is defined using OWL, and an approach for model-checking BPEL processes against PROPOLS properties using a transformation into finite state automata is presented.

[28] Weidlich et al. [WPDM10, WPD+11] seek to provide an intuitive set of metrics to quantify compliance. The approach is based on behavioral profiles. For example, compliance of logs to BP models can be evaluated.

[29] Schumm et al. & 2011 [DON+10, DDO+11] follow a pattern-based approach to compliance. Concretely, process fragments already validated as compliant can be brought

Table 3.1: Evaluation Notation of RCM Solutions

Symbol	Meaning
	Criterion is highly satisfied
	Criterion is satisfied
	Criterion is partially Satisfied
	Criterion is unsatisfied
	Criterion is highly Unsatisfied
	Criterion satisfaction could not be evaluated
	Evaluation criterion does not apply

together to form a new BP mode. However, it is trivial to see that nothing can guarantee that the obtained BP model will be compliant. This work is combined with [KKdV10] to model check the obtained BP models in order to evaluate compliance.

[30] Arsac et al. [ACPP11, ACKP11] employ model checking to validate BP models for compliance against security compliance requirements. The strength of this approach is that feedback is shown to the user after compliance checking that gives a concrete trace leading to a violation. However, as is often the limitation with model checking approaches, there is only a single trace leading to a violation and it is a state trace, not a task execution trace, so there is still a gap for the business user to understand the violation.

[31] Van der Aalst et al. [vdAdBvD05, Aal05, ADO+08, vdA06] use their process mining tool called ProM to collect event traces and reconstitute BP executions. These event traces are verified against LTL formulas expressing desired properties of the original BP model. This work is part of compliance audit, since compliance checking is conducted post-execution.

[32] Baldoni et al. [BBB+08] propose a domain specific language for curriculum design which allows to combine domain knowledge about the learner and pedagogical constraints. This language has a graphical concrete syntax and has formal semantics in LTL. It is in many ways similar to the work of Pesic [Pes08]. Verification of curricula is realized using model checking.

3.6 A Comparative Analysis of Solutions to RCM for BPM

This section gives a comparative evaluation of the selected RCM solutions. We use a set of symbols to score the approaches. The meaning of the symbols is given in Table 3.1. The approaches that are listed in Table 3.2 are assigned an ID following the pattern '[W-X]', where X is a number from 1 to 32, and accompanied with a single representative paper.

The evaluation criteria are divided into three categories. The first category concerns the components that are provided by the RCM solutions for assisting the business users: a modeling language, a methodology, an architecture describing how the solution is built out of existing software and how to extend it, tool support, and a repository for the CR models. The second category covers the eight functional areas of the RCM compliance dimension in the automated RCM conceptual framework (cf. Figure 3.1 in Section 3.3). Thirdly, we included a set of functional

and non-functional capabilities as evaluation criteria. In the following, we give a concise definition to each of the evaluation criteria appearing in the third category, in Sections 3.6.1 and 3.6.2.

3.6.1 End-User, Application and Violation Management

The *Multi-Formalism (Functional)* criterion refers to the ability of the framework to support several formalisms for expressing rules. Examples of such formalisms are OCL[7] for expressing constraints on metamodels (e.g., MOF[8] models), PRR[9] (an OMG standard for interchanging production rules) or temporal logics such as CTL[10] or ATL[11], etc. We refer the reader to [OA07, ETvdHP10a, PAN04, KMKP11b] for a discussion on the need to support multiple formalisms. The reasons for this are three-fold: (i) different styles in formal expression of constraints, (ii) different levels of tool support and different uses (e.g. model checking, simulation, etc.), and (iii) various formalisms needed to capture the semantics of CR constraints [KMKP11b]. This has been recognized by the business rules industry which has led for example to the introduction by the OMG of the PRR standard. Similarly, in the formal verification community the PSL[12] and PSP [MGJ98] languages for specification patterns have been introduced, which provide semantic mappings to various (e.g., temporal logic, regular expression) formalisms.

The *Formal Semantics (Functional)* criterion evaluates whether the compliance modeling language has full formal semantics or not. *Business User (BU) Orientation* refers to the set of non-functional criteria that judge whether the compliance framework is easy to use (e.g., easy specification of rules, easy modeling of CRs, user-friendliness of the language, push-button verification, conceptual clarity and adequacy, practicality for real-life cases, etc). Moreover, the modeled CRs should be linked to the impacted enterprise model elements (e.g., business process tasks, organizational roles) in some manner, and that is captured by the *Semantic Alignment* criterion.

Ideally, the approach used for managing RCM compliance could be used for different domains: business processes (classically referring in service oriented computing to service orchestrations), process choreographies, goal models, service architectures, etc. We call this capability support for *Multiple Application Domains*. The *Expressiveness* criterion judging the expressiveness of the language used for modeling CRs (e.g., rule, policy, ontology, logic) and is evaluated with regard to how much of the CR spectrum, i.e., the range of constraints and rules which might be expressed in a CR, can be covered using the provided language.

The *Weak Compliance* column indicates whether the solution distinguishes situations where the enterprise model is checked as compliant because the applicability condition of the CR is not fulfilled from the situation where it is really compliant. Additionally, the ability of the RCM solution to provide the cause(s) (e.g., a sequence of events) of each violation and possibly a localization of the violation cause(s) in the EM is shown by the *Explanation / Localization* criterion (cf. [Awa10]).

[7] Object Constraint Language
[8] http://www.omg.org/mof/
[9] Production Rule Representation
[10] Computation Tree Logic
[11] Alternating-time Temporal Logic
[12] http://www.eda.org/ieee-1850/

Table 3.2: Comparison of RCM Solutions along RCM Framework Evaluation Criteria

Approach		Solution Components					RCM Framework Alignment								Functional & Non-Functional Capabilities																			
															End-User			Application				Violation			Rule Class			Business Aspect				Powers	
ID	Representative Paper	Language	Method	Architecture	Tool	Repository	Modeling	Verification	Enforcement	Monitoring	Audit	Consistency Checking	Conflict Resolution	Reporting	Violation Management	Multi-Formalism	Formal Semantics	BU Orientation	Semantic Alignment	Multi-Application Domain	Expressiveness	Weak Compliance	Handling	Explanation/Localization	Reparation	Structural Rules	Temporal Rules	Contractual Rules	Data	Resource/Task	Human/Role	Time Deadlines	Delegation	Revocation
[W-1]	[Awa10]	✓	✓	✓	✓	◆	✓	✓	✗	✗	✗	✓	✗	✗	◆	◆	✓	◆	✓	✗	✓	✓	✗	✓	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗
[W-2]	[LMX07]	✓	✓	✓	✓	◆	✓	✓	✗	✗	✗	✗	✗	✗	✗	◆	✓	◆	✓	✗	✓	✓	✗	✗	✗	✓	✓	✗	◆	✗	✗	✗	✗	✗
[W-3]	[LRMGD09]	✗	✗	✗	✓	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✓	◆	✗	✗	✗	✗	✗	✗	✗	◆	✗	✗	✗	✗	✗	✗	✗	✗
[W-4]	[GM05]	✓	✗	✗	✓	✗	✓	✓	✗	✗	✗	✓	✓	◆	◆	✗	✓	✗	✓	✗	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗	◆	✗	✗
[W-5]	[EKP09]	✓	✗	✓	◆	✓	✓	✓	✗	✗	✗	◆	✗	✗	✗	✗	✓	✗	✓	◆	✗	✗	✗	✗	✗	✓	✗	✗	◆	✗	◆	✗	✗	✗
[W-6]	[Web09]	✗	✗	◆	✓	✗	✓	✓	✗	✗	✗	⚠	⚠	✗	✗	✗	✓	◆	◆	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
[W-7]	[Nam08]	✗	✓	✓	✓	✓	✓	✓	✓	✗	✗	◆	✗	✗	✓	✗	✗	◆	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
[W-8]	[RCR07]	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	◆	✗	✗	✗	✗	✓	◆	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
[W-9]	[ETvdHP10b]	✓	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	◆	◆	✗	✗	◆	✗	✗	◆	◆	◆	✓	✗	✗	✗	◆	✗	✗	✗
[W-10]	[KA09]	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	◆	✗	✗	✗	✗	✓	✗	◆	✗	✗	✗	✗	✗	✗
[W-11]	[GV06]	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	◆	◆	✗	✗	✗	✗	◆	◆	◆	✗	✗	✗	✓	✗	✗
[W-12]	[GK07]	✗	✗	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗	✗	◆	◆	✗	◆	✗	✗	✗	✗	✗	✗	✓	◆	◆	✗	✗	✗	✗	✗	✗	✗
[W-13]	[GLM+05]	✓	✓	✗	✗	✗	✓	◆	◆	◆	◆	✗	✗	✗	✗	✗	✓	◆	✗	✗	✓	✗	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓	✗	✗
[W-14]	[Pes08]	✓	✓	✓	✓	✗	✓	◆	✗	✗	✓	✗	✗	✗	✗	✗	✓	◆	✓	◆	✗	✗	◆	✗	✗	◆	✓	✗	◆	◆	✓	✗	✗	✗
[W-15]	[FPR07]	✗	◆	◆	✗	◆	✓	✓	✗	✗	✗	◆	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✓	◆	✓	✗	✗	✗
[W-16]	[PGS+05]	◆	✗	✗	✗	✗	✓	?	✓	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
[W-17]	[FESS07]	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	◆	◆	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
[W-18]	[KL08]	✓	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	◆	✓	✗	◆	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗
[W-19]	[WSM08]	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	◆	✓	✓	✓	✗	✗	✓	✗	✗	✗	✓✗	✗	✓	✓	✓	✗	✗	✗
[W-20]	[RCJL06]	✗	✗	✓	✗	✗	✗	✗	✓	✓	✓	⚠	⚠	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	◆	✗	✗	✗	✗	✗	✗	✗
[W-21]	[Uni08]	◆	✓	✓	✓	✓	✓	✓	✗	✓	?	?	?	✓	◆	✗	✓	✓	✓	✗	◆	?	?	?	?	?	✓	✗	?	✗	✗	✗	✗	✗
[W-22]	[SM02]	✗	✗	✗	✓	✗	✓	✓	✗	✗	✗	◆	✗	✗	✗	✗	✓	✗	✗	◆	✓	✗	✗	◆	✗	◆	◆	✗	✗	✗	✓	✗	✓	✓
[W-23]	[SLS06]	◆	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	◆	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
[W-24]	[SAD+10]	✓	✓	✗	✓	✓	◆	✗	✗	✗	✗	✗	✗	◆	✗	✗	◆	✓	✓	✗	✗	?	✗	◆	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[W-25]	[DNS08]	✓	✓	✓	✗	✓	✓	◆	✗	✗	✗	◆	✗	✗	✗	✗	✓	✗	✗	◆	✓	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗
[W-26]	[JMKH07]	✓	✗	✗	✓	✗	◆	◆	✗	✗	✗	⚠	✗	✗	✗	✗	✓	◆	✗	✗	✗	✗	✗	✗	✗	◆	◆	✗	◆	✗	✗	✗	✗	✗
[W-27]	[YMH+06]	✓	✗	◆	✓	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	◆	✗	✗	◆	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
[W-28]	[WPD+11]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	⚠	⚠	✗	✗	✗	✓	◆	⚠	✗	✗	✗	✗	◆	✗	◆	✗	✗	✗	✗	✗	✗	✗	✗
[W-29]	[SLM+10]	✓	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	◆	✓	✗	◆	✗	✗	✗	✗	◆	◆	✗	◆	✗	✗	✗	✗	✗
[W-30]	[ACPP11]	✓	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	◆	✗	✗	✗	✗	✗	✗	◆	✓	✗	✗	✓	✓	✓	✗	✗	✗
[W-31]	[ADO+08]	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	◆	✗	✗	◆	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
[W-32]	[BBB+08]	✓	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	◆	✗	✗	◆	✗	✓	✗	✗	◆	✗	✗	✗	✗	✗

Generally, some RCM frameworks are not only capable of discovering a violation, but to automatically undertake *Recovery* actions. Approaches fulfilling this criterion can allow the user to define corrective action upon occurrence of a violation such as notifications or changes to the BP. The first kind of *Recovery* is *Handling*, which we define as the ability of some RCM frameworks to allow to model how to react to violations by triggering actions (cf. *Handling* column) that do not alter the BP. An example is the logging of information about the violation. The other kind of *Recovery* is *Reparation* which is the ability of the solution to react to violations by triggering repairing action that will modify the model so that the violation does not occur again (e.g., as in [Nam08]). This can be semi-automated so as not to require human intervention each time a violation occurs by for instance, specifying standard changes to process models.

Compensation is another kind of *recovery* that was not included in our evaluation in Table 3.2. A *Compensation* consists of the enforcement of actions that correct the state of the BP so that the violation effects are eliminated. A typical example is that when a medicine leads to health damage for the people who use it, the chemical laboratory producing the medicine is liable to pay a fine to the authorities and to the victims.

3.6.2 Rule Language Expressiveness, Business Aspects and Policy Powers

In the approaches we surveyed, and conforming to [KMKP11b], we distinguish three kinds of rules that are supported by CR modeling languages. *Structural Rules* express invariants over the elements of the BP model (e.g., data objects, task properties, human worker properties). *Temporal Rules* express dependencies between individual tasks composing a BP model, while *Contractual Rules* Express deontic constraints (e.g., duties, rights) on objects defined in the EM. A CR modeling language may allow to express CRs in one or several of these kinds of rules. The REALM framework [GLM+05] for example, combines structural constraints on objects as well as temporal constraints over moments in time at which the structural constraints must be true. The following constraint is structural and temporal and is adapted from a bank example from [GLM+05]:

$$
\begin{aligned}
&\forall bank \in Bank, acc \in Account, retain \in Retain : \\
&\exists close \in Close \ / \\
&\Box t_{delete}.DoOnF(bank, retain, acc.customer.record) \\
&\rightarrow (\blacklozenge t_{close}.DoOnF(bank, close, acc) \\
&\wedge t_{delete} - t_{close} \geqslant 5_{[year]})
\end{aligned}
$$

This formal expression of a CR basically says that a customer account record may only be deleted if the account has been closed for at least 5 years.

The other dimension on which to evaluate the expressiveness of CR modeling languages is the *Business Aspect* dimension. It is named like this to distinguish it from the CRs which only concern the control flow (i.e., the ordering of the tasks in the BP model), without considering the environment (i.e., the enterprise model) in which the BP model is executed. Several aspects are usually of interest. *Data* constraints are expressed on data values, such as data integrity rules or simple data-based derivation rules. See [PAN04] for a detailed discussion of the different uses of business rules.

Other approaches tackle CRs that express constraints on the usage of resources by the BP or on the allocation of tasks to humans or applications (agents) responsible for their execution. That is what the *Resource/Task & Human/Role* criteria deal with. Usually, access control CRs belong to this category, and approaches are distinguished by the richness of the (domain-specific) vocabulary they make available for expressing constraints. Separation of duty is also an exam-

ple of application of such constraints. We add to the latter criteria the support for *Time* deadlines.

A separate category of functionalities of RCM frameworks are *Policy Powers*, which are policies that act on other policies. Typical examples are *Delegation* and *Revocation* of permissions and obligations. The approaches providing this kind of functionality usually come from the distributed systems security community where various logics for expressing powers exist, but to our best knowledge of the literature, apart from some exceptions such as [Gaa10], they have not been much applied to RCM yet. Under these criteria, and for the purpose of evaluation, we also included other types of policies such as the delegation of tasks in a BP or the delegation of role in a BP. The opposite function of delegating is the revocation of a granted (respectively delegated) permission or obligation.

3.7 Discussion

From the comparative analysis made in Table 3.2, we retained the following solutions as representative for very promising directions of work, not only in the scope of this thesis, but for tackling the RCM for BPM problem in general. The list is given here using the ID of each stream of work, the reader will find the corresponding references in Table 3.2: [W-1] (Awad et al.), [W-4] (by Governatori et al.), [W-2] (Liu et al.), [W-9] (Elgammal et al.), [W-27] (Yu e al.), [W13] (by IBM), [W-14] (by Pesic et al.) and [W-19] (by Wolter et al.). Although we did not build our thesis contribution on these as we pursue a different objective, these works and all the works surveyed certainly have enriched our understanding of the RCM problem and its challenges, thus contributing indirectly to this thesis.

The work [W-1] [Awa10] is in our impartial opinion one of the most advanced and contains several useful ideas for a comprehensive RCM solution. For BPMN modeling, although the author claims generality of his approach, we think [W-1] work is valuable. For [W-2] [LMX07], a useful output of this research is the BPSL language, which is a different approach from the PSP patterns as well as the set of optimizations borrowed from model checking. Adapting these ideas to more business-oriented BP modeling languages, e.g., including resource usage or role-based constraints, would certainly be a direction of work with impact.

The work stream [W-4] in [GMS06, LSG08, SG10, GR10a, LSG07, Gov05, KGS10] which is based on FCL/PCL is interesting from a violation management point of view, where it seems the most advanced. It allows to reason non-monotonically, which is a major advantage over many of the other works. Also, this stream of work describes a full-fledged logic system, including a proof system and mechanisms to check consistency and normalize the set of rules written in this formalism. FCL deals natively with all kinds of obligations and a comprehensive ontology of obligation types with different semantics was developed by this stream of work. For example, it is the only work we know of in the field of BPM compliance which describes pre-emptive and non-preemptive or achievement and maintenance obligations [GR10b]. It also defines several measures of compliance depending for example on whether the compliance is achieved during at least one process execution but not all, or if compliance is ensured over all possible execution traces. The language would profit from a concrete syntax more amenable to BUs than the logic-inspired concrete syntax provided by

SPINdle. Another limitation of the approach is that it relies on (partly manual) annotations of the process tasks with propositional statements, which requires some effort by the process modeler.

Regarding [W-9] [ETvdHP10b], the pattern-based and graphical concrete syntax compiles into LTL statements, the introduction of a *contrary to duty (CTD)* operator for describing violation compensation actions, the modeling tool are all features that make this work very useful for leveraging CR specification (modeling) to the level of business users. The CTD operator (denoted by the $\otimes$ symbol) is a non-monotonic logical operator on actions defined in the FCL logic in [W-4] [GM05]. It is used to define a prioritized chain of actions to be enforced upon the non-fulfillment of an action obligation. In [ETvdHP10b] however, it is used with different semantics[13].

PROPOLS [YMH+06] in [W-27] is in the same line of work as [W-9]; this work goes a step further into making the specification of temporal CRs easier and abstract away form the underlying temporal logic semantics. It lacks however, just like [W-9], several of the elements required for a comprehensive RCM solution (e.g., business aspects, cf. Table 3.2).

In [W-13] [GLM+05], the REALM framework allows the integration of business models in the specification of CRs as well as the management of regulation as part of RCM. These are two distinctive aspects of the work which make it highly interesting for a holistic view on RCM that considers the business environment of a BP.

Although not targeted at the same problem as the main stream of work, the declarative modeling approach of the solution in [W-14] [Pes08] (graphical concrete syntax and pattern-based CRs) which regard not only temporal aspects of CRs but also organizational ones (used in structural CRs) make it very interesting. For the class of RCM problems where a declarative specification (e.g., for requirements validation, mock-up design purposes) is suitable, we consider this to be a very inspiring solution. We see much synergy between this work and ours, in what concerns the specification of temporal constraints.

Finally, we retained [W-29] [DON+10] as this technique employs both the classical *divide and conquer* and the *abstraction* strategies to deal with the complexity of modeling and integrating CRs in BPs. Compliant BP model fragments can be regarded as templates or building blocks helping the expert BP modeler build process models which are 'locally compliant' in a quicker manner. This approach likely has a high potential in easing the burden of business users involved in RCM tasks, as business users can direct their attention away from the overall regulation towards smaller chunks of functionality that is provably compliant as represented by BP fragments. However, one may express concerns about this approach as no statement about the compliance status of a BP model built out of BP fragments can be made. In general, the assertion that if a set of process blocks are each individually compliant then the whole process which these blocks collectively form (block-based process composition) is compliant is not true. Ultimately, the complexity of checking compliance of the whole process is

[13] To our best understanding, in [ETvdHP10b]: $\mathrm{CR}_1 \otimes \mathrm{CR}_2$ expresses an alternative CR_2 to fulfill in case of a violation of a CR_1 (i.e., it is equivalent to: $\mathrm{CR}_1 \vee (\neg \mathrm{CR}_1 \wedge \mathrm{CR}_2)$) but is not adequate for non-monotonic compliance requirements. Compliance requirements require non-monotonicity among other reasons in case of the need for compensations. For instance, by extending the semantics of the classical boolean $\vee$ operator with time sequence semantics the expression $CR_1 \vee CR_2$, is not equivalent to $CR_2 \vee CR_1$, since the order of fulfillment of compliance requirements matters. In other words, the obligation of fulfilling CR_2 does not exist until the obligation of fulfilling CR_1 exists and it is violated.

retained. It is interesting to look for classes of BP compliance problems for which this approach is fitting, i.e., the complexity linked to checking the block-based process composition is acceptable.

We insist on the fact that the reader may make his own selection of approaches, based on his appreciation of which evaluation criteria are most important. In our case, for RCM, we consider that business users are the primary end users of an RCM solution. Therefore, the language for modeling compliance should have a formal foundation in order to allow for transformations and checking, but should also allow non-logic experts to use the language easily. We noticed in several works that attempts using compliance requirement templates and patterns or compliant BP fragments could be linked to this objective. Another aspect guiding our evaluation is the fact that in our view, RCM should not only cover control flow, workflow or data flow, but also additional aspects such as resource usage, organizational entity properties (e.g., roles or location), goals, risks, policies and rules, etc. This is why approaches with a holistic perspective on RCM might have better adoption chances.

3.8 Summary

Regulatory compliance management (RCM) is attracting much interest from academia, especially in recent years. However, grasping RCM is still hard for newcomers to this topic of research. The chapter contributes a conceptual framework for understanding the problem of RCM in BPM and navigating through the existing research. We realized a comprehensive survey of existing solutions to RCM, and extracted a battery of solution evaluation criteria by looking at the strengths and weaknesses of every solution. This enabled us to do a comparative analysis of the surveyed solutions along the elicited criteria. We found that some approaches are very advanced and tackle many of the RCM solution criteria. We also have found that supporting business users in modeling compliance requirements and a complete integration into business process-centered enterprise models is not yet satisfactory and needs further research. The goal of this thesis is to contribute to this line of research.

Part II

CoReL Framework for Regulatory Compliance

Chapter 4

The System Model

There are two ways of constructing a software design: one way is to make it so simple there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.

Tony Hoare.

We refer by system to the abstraction used in this thesis on a business process-centered enterprise model. This means that a system is a representation of an enterprise information system which is dynamic, i.e., can execute actions. In this chapter, we will give a definition of a system as a combination of two elementary concepts: ASE Triples, which are introduced in Section 4.1, and a special class of block-based business processes we will name SBP (which stands for Structured Business Processes) introduced in Section 4.2. Section 4.3 presents the metamodel of SBP and Section 4.4 explains its formal semantics.

4.1 The ASE Model

The ASE model describes the basic interaction entities in a system. It is composed of the triple: Action, Subject, Entity. This is illustrated in Figure 4.1.

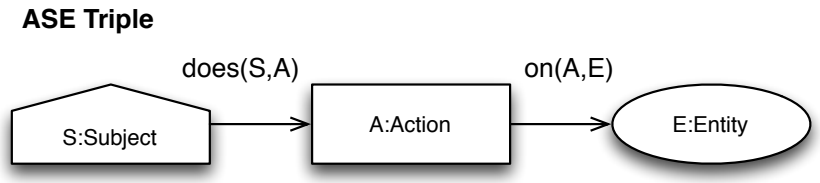


Figure 4.1: ASE Triple

In the basic ASE model, we consider the following four algebraic sorts. First, we partition the system into two sorts: Actions and Objects.

- Actions (A): Represents the operations that are supported by the system.
- Objects (O): These operations are executed by and on elements of the Objects sort called Objects. This sort has two non disjoint sub-sorts Subjects and Entities.
  - Subjects (S): it represents the system objects capable of executing actions.
  - Entities (E): it contains all system objects on which actions may be executed.

Using these sorts, we define the following predicates:

- Does(s,a): means that the subject s executes action a. This predicate does not specify on what entity the action is executed.
- On(a,e): means that action a is being executed on some entity e. The subject is not specified by this predicate.
- Doing(a,s,e): means that subject s is currently executing action a on entity e. We have the following equivalence: $Doing(a, s, e) \triangleq Does(s, a)$ && $On(a, e)$.

We use these predicates to explain the most elementary constructs on the ASE system model. In Section 4.2, we show how we use ASE triples to define a basic formal business process language in Alloy.

ASE Triple in the System Model

In the context of our work, we view system models as behavioral constellations where the basic interaction entities are ASE triples. If we take the history of events during the lifetime of a system, we can represent it as a sequence of ASE triple executions. In plain english, this means that a system's behavior is simply the (possibly infinite) sequence of events that each correspond to the execution of an ASE Triple. here, we purposely ignore the time dimension in the occurence of action events. This idea is illustrated in Figure 4.2

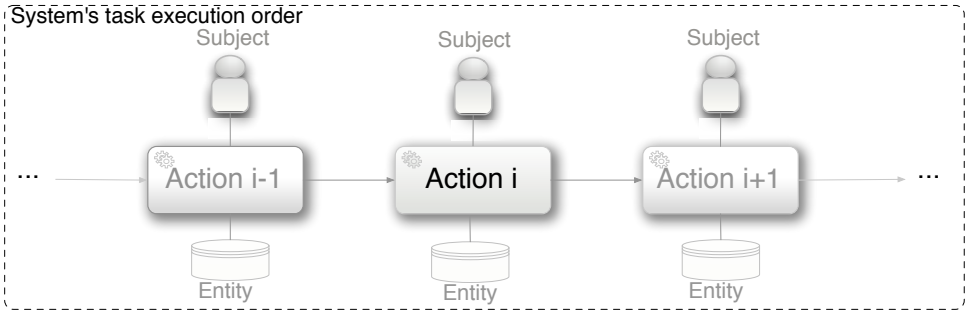


Figure 4.2: System Execution Model

While Figure 4.2 shows the global event history perspective on our view of a system's behavior, we can further refine this view. We do this along the ASE paradigm, which distinguishes between three atomic states relative to the ASE triple, as shown in Figure 4.3:

- The ASE triple is ready to be executed but no yet executed. The predicate which becomes true in this state is: Do(a_i,s_i,e_i).
- The ASE triple is being currently executed: Doing(a_i,s_i,e_i).
- The execution of the ASE triple is finished: Done(a_i,s_i,e_i).

The reader may notice that our model suggests that the following causal relation exists: $Done(a_{i-1}, s_{i-1}, e_{i-1}) \Rightarrow Do(a_i, s_i, e_i)$. In practice, system models can be of a variety of types. In this chapter, we will define a business process modeling language, which implements our ASE paradigm. We first introduce a block-based process modeling language in the next section 4.2, which we then extend with ASE triples in the following section.

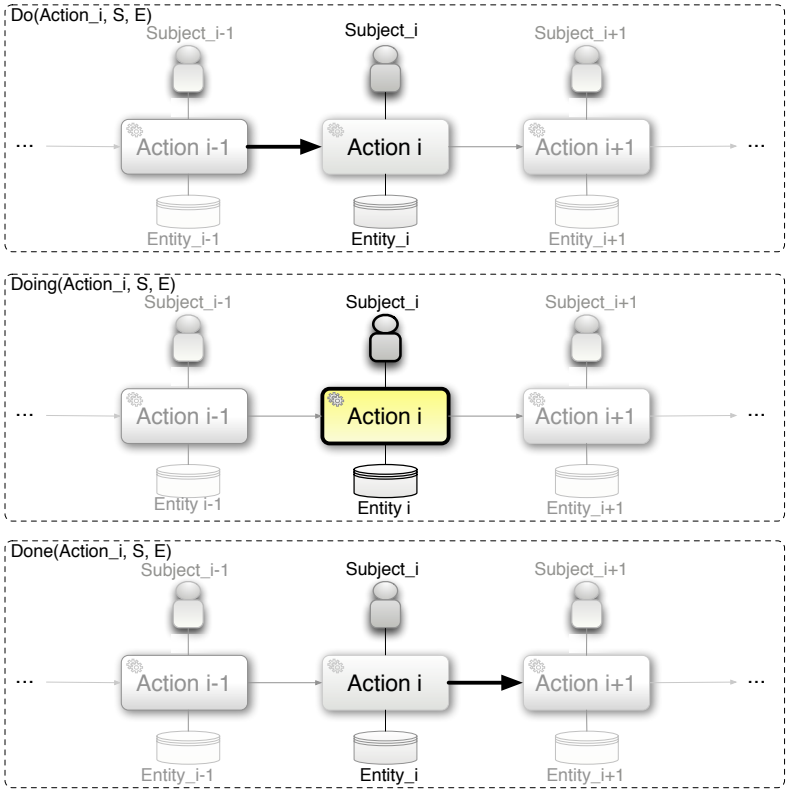


Figure 4.3: Do | Doing | Done Predicates in System Execution

4.2 The SBP Model

We first of all give some background definitions of a small business process modeling language which is a subset of a comprehensive standard process modeling languages. The definition of this language was introduced in previous shared work related to a formalization of compliance management of obligations [CTEKG+13]. Then we show how we can build the definition of SBP models on top of that. We finish this subsection with a small example of an SBP model.

4.2.1 Background Definitions

In our approach, a process is composed of tasks and control flow rules. Tasks represent the system actions that are done during the execution of a process. Control flow rules are used to define the valid routings (or flows) to traverse a process model, thereby defining valid execution traces of such a process model.

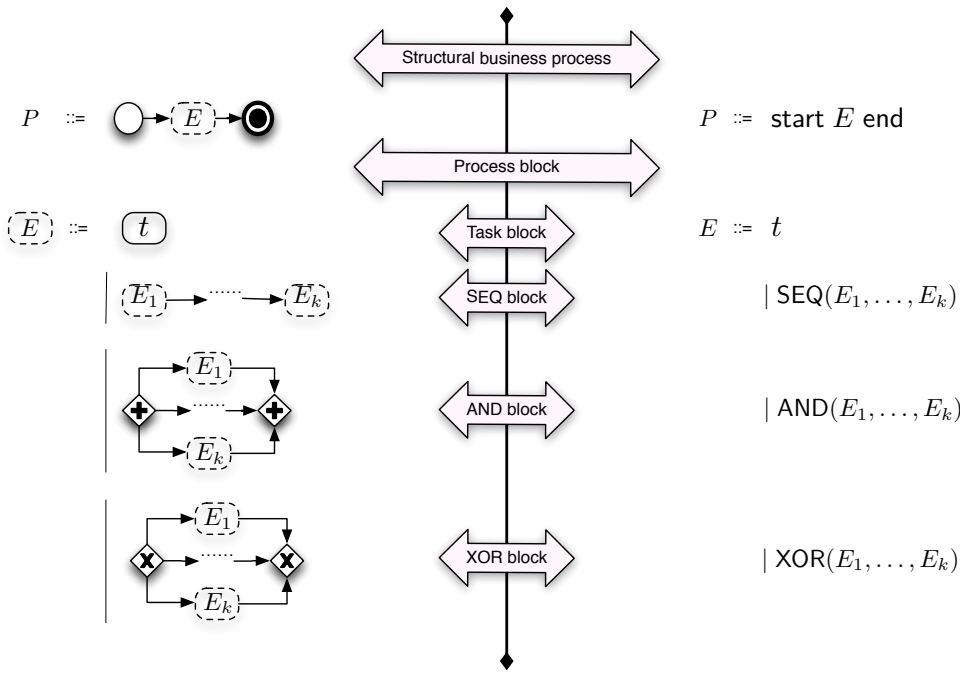
For instance control flow rules can define that a certain task has to be followed by another one, that two tasks are executed concurrently or are mutually exclusive. Arrows connecting the elements of a process identify a general process task execution order in which the elements can be executed.

To represent a process we use a fragment of BPMN[1], a well-established business process modeling notation. The fragment considered uses only *AND* and *XOR* control flow rule nodes in addition to the *start* and *end* nodes. The *AND* control flow rule is used to coordinate tasks which can be executed concurrently. The *XOR* is used to model exclusive choice in the execution of a set of tasks. For each of the two control flow rule types we consider, *AND* and *XOR*, we distinguish between split and join node subtypes. Split nodes signal the start of the concurrent/exclusive execution of the enclosed tasks, while Join nodes signal the end of the execution of the enclosed tasks.

A process is structured if it consists of hierarchically nested blocks as depicted by Definition 4.2.1. Previous works have already considered such structures as in [BKB00] and [PGBD12]. While not all processes are structured, structured processes are a substantial class of real-life processes. According to [PGBD12] 406 of the 604 ($\approx$ 67%) processes in the SAP reference models [KT98] are structured. In addition [PGBD12] identifies conditions under which unstructured processes can be transformed into structured ones, and proposes an algorithm for the transformation. They report that 78 of the unstructured processes in the SAP reference model can be converted into behaviorally equivalent structured process models. This raises the percentage of reference business process models which can be represented as structured processes to $\approx$ 80%.

This is obviously an anecdotal argument, but the postulate behind this is that we can reasonably assume that a significant portion of used business processes can be represented as structured business processes. We work with this assumption in the rest of this thesis and focus our research on structured business processes.

Definition 4.2.1 (Process)**.** *A structured business process P is a business process generated by the following grammar given in the format of a graphical extension of BNF (with the vertical lines indicating alternative for the right hand side):*



The control flow rule ◯ is called *start* and the control flow rule ◉ is called *end*. The control flow rule ✚ is called *AND split* in case of multiple outgoing arrows and *AND join* in case of

[1]Business Process Model and Notation, Version 2.0, http://www.omg.org/spec/BPMN/2.0

multiple incoming arrows. A pair of *AND split* and *AND join* control flow rules groups a set of sub-blocks indicating a logical relationship to activate all the sub-blocks concurrently. Finally, the control flow rule  is called *XOR split* in case of multiple outgoing arrows and *XOR join* in case of multiple incoming arrows. A pair of *XOR split* and *XOR join* control flow rules groups a set of sub-blocks indicating a logical relationship to activate exactly one of the sub-blocks, chosen arbitrarily.

We assume that all the tasks in a structured business process carry a distinct identity that constitutes a key part of the label of a task. Therefore, a task (t) can directly be referenced by its label t. Similarly, (process) block identities are also distinct hence a block (E) can directly be referenced by its label E. As a consequence, for simplicity, we also allow a textual way to reference the graphical representation of structured business processes.

Example 4.2.1. *In Fig. 4.4 we provide an example of a process containing four tasks labeled* $t_1, \ldots, t_4$. *Within the process is shown an* XOR *block containing in different branches the tasks* t_1 *and* t_2. *The* XOR *block is nested within an* AND *block, forming one of its branches and task* t_3 *forming the other one. The* AND *block is preceded by the* start *coordinator and followed by task* t_4 *which in turn is followed by the* end *coordinator.*

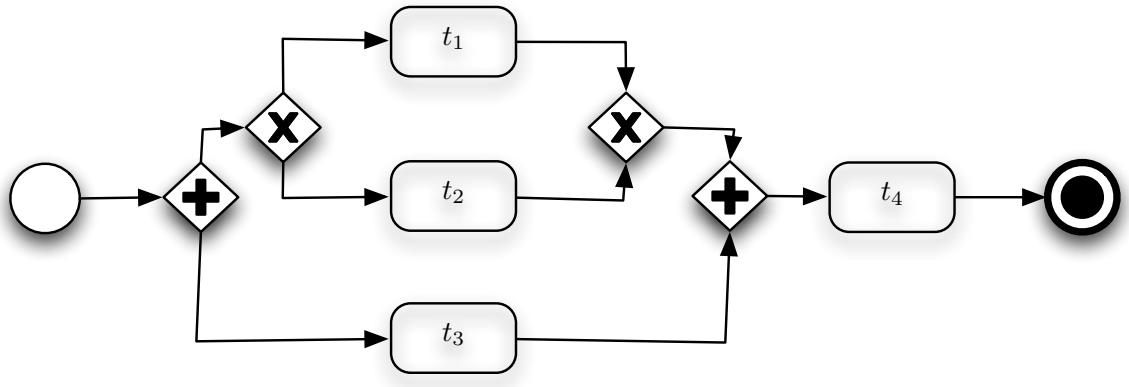


Figure 4.4: Process Example

Given a process modeling an activity, an execution of such a process represents one way to perform it. An execution is a valid serialization of a subset of tasks composing the process. A serialization is considered valid if it starts from the *start* coordinator and terminates at the *end*. In addition a valid serialization has to comply with the semantics of the coordinators and the connections between the tasks.

A process is defined as $P = \mathit{start}\ E\ \mathit{end}$. An execution of P is equivalent to executing the block E within *start* and *end*. Thus we will provide the formal semantics for executing blocks which can be used for process execution as well.

Definition 4.2.2 (Block Execution)**.** *A process block* E *can be serialized into a set of finite sequences of tasks, written* $\Sigma(E)$*, defined by the following structural recursion. We call each sequence in* $\Sigma(E)$ *an execution of* E*, ranged over by* ε.

1. $E = t$: $\Sigma(E) = \{(t)\}$;
2. $E = SEQ(E_1, \ldots, E_k)$: $\Sigma(E) = \{\varepsilon_1; \ldots; \varepsilon_k \mid \varepsilon_1 \in \Sigma(E_1), \ldots, \varepsilon_k \in \Sigma(E_k)\}$, *where* ; *stands for sequence concatenation.*

3. $E = XOR(E_1, \ldots, E_k)$*:* $\Sigma(E) = \Sigma(E_1) \cup \ldots \cup \Sigma(E_k)$*;*

4. $E = AND(E_1, \ldots, E_k)$*:* $\Sigma(E) = \{(t_1, \ldots, t_n)\}$ *such that*

(a) $\forall i, 1 \leq i \leq k, \exists \varepsilon_i \in \Sigma(E_i)$ *such that* $\{t_1, \ldots, t_n\} = \bigcup_{1 \leq i \leq k} Tasks(\varepsilon_i)$

(b) $\forall E_i \in E = AND(E_1, \ldots, E_k), t_h, t_j \in E_i | t_h < t_j \rightarrow \forall \varepsilon \in \Sigma(E), t_h > t_j$.

Namely $\Sigma(E)$ *is the set of sequences each of which merges a sequence of* $\Sigma(E_1)$*, ..., and of* $\Sigma(E_k)$*. Merging a set of sequences gives rise to a sequence that includes all the elements of the operand sequences. Moreover, the ordering in the result sequence should be compatible with the ordering in the operand sequences.*

A process conform with Definition 4.2.1 contains only tasks that can be executed. This means that a process cannot contain a task that does not belong to any of it possible executions.

Example 4.2.2. *Take into account the process in Fig. 4.4 as* P = *start* E *end. We have that* $\Sigma(E) = \{\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4\}$ *where* $\varepsilon_1 = (t_1, t_3, t_4), \varepsilon_2 = (t_2, t_3, t_4), \varepsilon_3 = (t_3, t_1, t_4)$ *and* $\varepsilon_4 = (t_3, t_2, t_4)$*.* $\Sigma(E)$ *contains the four possible executions of the process* P*. An execution not contained in* $\Sigma(E)$*, like* $\varepsilon_5 = (t_3, t_4, t_1)$*, is not a valid execution of* P*. In this particular case one of the reasons why* ε_5 *is not a valid execution is because after* t_4 *the task* t_1 *is executed, which is not possible because* t_1 *belongs to an* XOR *block nested in an* AND *block that precedes the task* t_4 *in a sequence block.*

4.2.2 Example: Printer

We show in this section how to model a simple printer management example. This example shows how the constructs introduced earlier are combined to model a concrete business process. The process is shown in Figure 4.5, while the underlying block structure is shown in Figure 4.6.

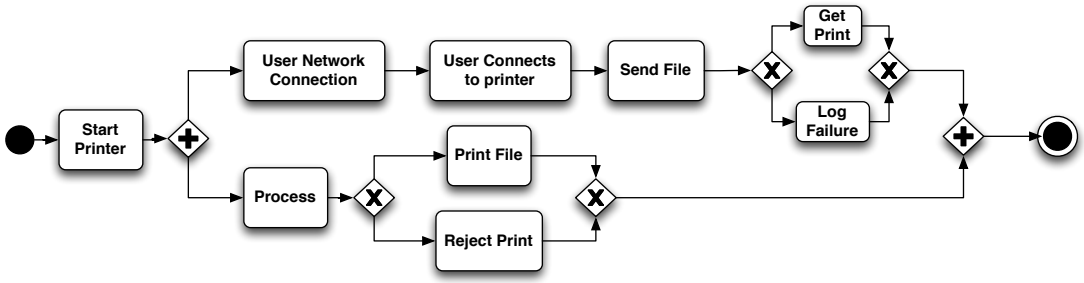


Figure 4.5: Simple Printer Example

In Figure 4.6, we see the six blocks making out the process from Figure 4.5. For pure illustration purposes, the Start node is this time displayed with a green 'play' button, while the End node is displayed with a red 'stop' button. The six interleaved blocks are numbered from the outermost block B1 to the innermost blocks B4 and B6 by decreasing containment order. Block B1 is a SEQ (for sequence) block containing task *Start Printer* and block B2. Block B2 is an AND block, B3 and B5 are sequence blocks, while B4 and B6 are both XOR blocks.

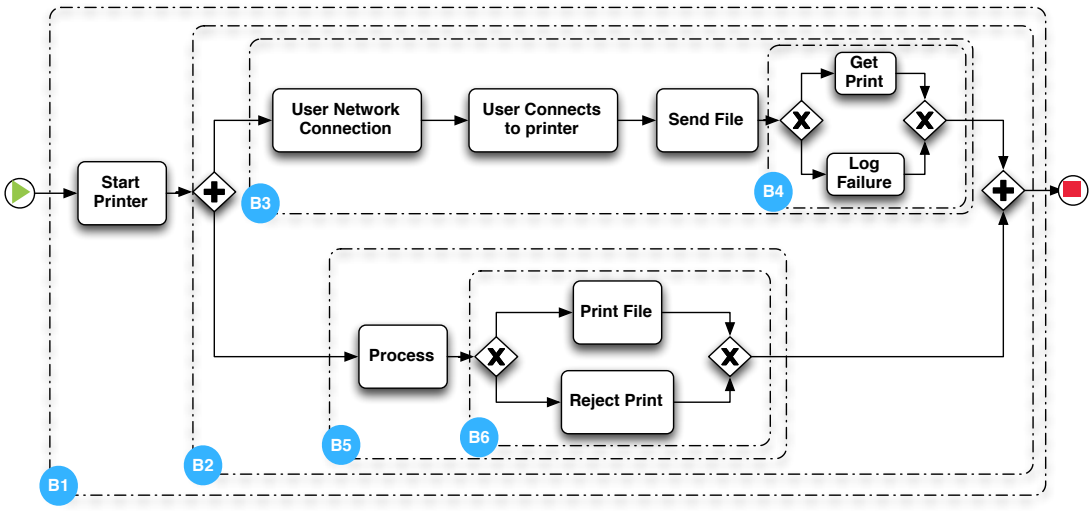


Figure 4.6: Simple Printer Example - Block Breakdown

4.3 Formalizing the System Abstract Syntax Metamodel

The business domain of application for our framework, which we call here the system, are business process-centered enterprise models. These are defined as the composition of SBP (block-based workflows) with ASEs. the basic intuition behind the relationship between SBPs and ASEs is that SBPs define a set of possible system executions where the atoms are the events corresponding to the execution of a given ASE. The information required to model a given ASE is usually available in standard process modeling languages such as BPMN, e.g., by using process pools or lanes to specify the Subject in an ASE. In this section, we define the Alloy constructs necessary to model these business processes called SBPs, using a metamodel. SBP stands for Structured Business Process. Building on this metamodel and its formalization in Alloy, we will later define the state-based operational semantics of its execution using Alloy.

ASE Triples

The metamodel we base our Alloy description on is given in Figure 4.7. We first create a new module called *ase*, which we need to implement the ASE triples in Alloy. We then define the following signatures: Action, Object, Subject and Entity. Later, when we will use Alloy on real examples, we will need to turn the Action, Subject and Entity signatures into abstract ones, and extend them with concrete signatures corresponding, for example, to concrete actions in the example.

We also add an abstract signature called qualifier, which is used to define relations of objects to properties. We will detail the use of this construct later in the thesis, and show why it is useful for verifying richer compliance requirements. We finally define the abstract signature *ASE* which is related to at most one action, subject and entity.

```
75 module ase
76
77 sig Action{}
```

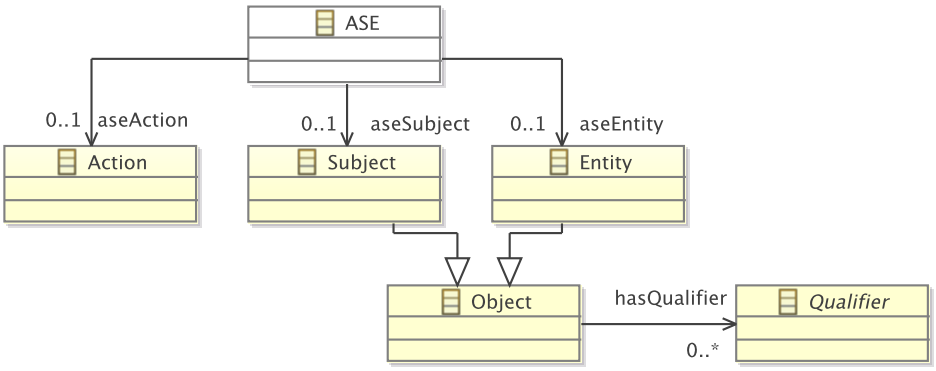


Figure 4.7: The ASE Metamodel

```
78  {
79    some se:ASE| this in se.a
80  }
81
82  sig Object{
83    q: some qualifier
84  }
85
86  abstract sig Subject extends Object{} {
87    some a:ASE| this in a.s
88  }
89
90  sig Entity extends Object{} {
91    some a:ASE| this in a.e
92  }
93
94  abstract sig qualifier{}
95
96  abstract sig ASE{
97    a: lone Action,
98    s: lone Subject,
99    e: lone Entity,
100 }
```

Modeling the Business Domain

For the rest of this subsection, we will base our Alloy description on the metamodel given in Figure 4.8. The Alloy code is informally explained throughout this document, so that it is sufficient to read the explanation text to understand the formalization. We first define a module called *sbp*, for structured business processes. We then import the *ase* module where ASEs are defined. Each part of the metamodel's formalization is explained in a separate paragraph.

ASE Triples in SBPs

In this part we define the abstract signature *Node* and the signature *AseNode*. An ASENode is a Node which acts as as a place holder for an *ASE* triple (named *se* in the signature relation).

```
101 -- ******Nodes *******
```

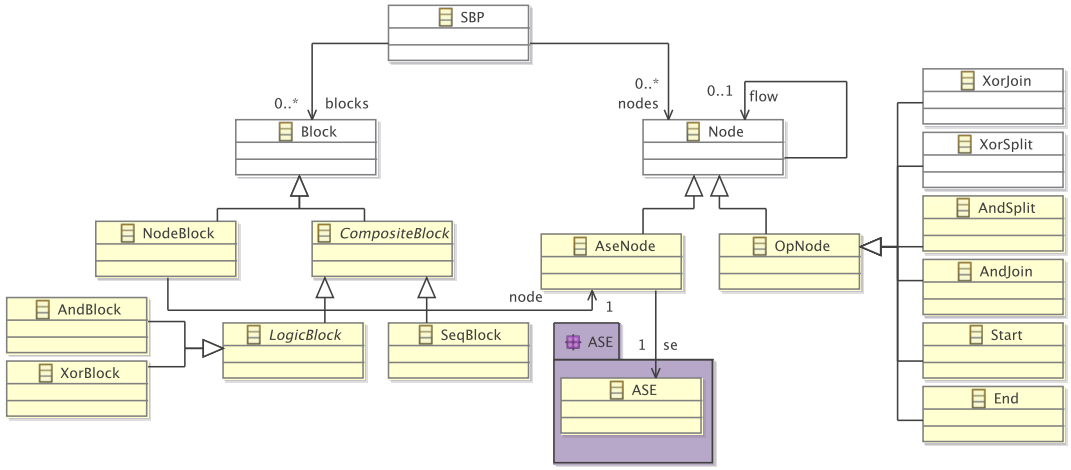


Figure 4.8: The SBP Metamodel

```
102 abstract sig Node{}
103 sig AseNode extends Node {
104   se: ASE
105 }
```

Consistency Constraints on ASE Nodes

We define two constraints on ASES. The *noAseSharing* fact says that no two *ASEs* might be shared by two *AseNodes*. This constraint is not required in real life business processes since we might have situations where the same *ASE* is used several times in the same business process. We use this Alloy fact because we want to have model instances generated which contain a given set of *ASEs* while having a minimal size. If we allow for redundant occurrence of *ASEs* in *AseNodes* we will not have that anymore.

```
106 fact noAseSharing {
107   all n1,n2: AseNode| n1·se ≠ n2·se
108 }
109
110 fact noOrphanAse {
111   all a:ASE| one n: AseNode| n·se = a
112 }
```

The second constraint is modeled by the fact *noOrphanAse* which expresses the consistency constraint that we shall not have *ASEs* not contained in a *AseNode* (e.g., orphan or floating ASEs). Thereby, we guarantee that all *ASEs* are part of the business process workflow.

SBPs as Directed Acyclic Graphs of Nodes

We build an *sbp* as a directed acyclic graph (DAG) whose nodes are *ase*s. In Alloy this can be defined elegantly by using the generic *graph* library and parametrizing it with the *ase* signature.

```
113 module sbp
114
115 open ase
116 open util/graph[Node] as gase
117
```

```
118 abstract one sig SBP{
119   flow: Node →Node,
120 }{
121   // A structured business process (system) is a DAG of ases
122   // graph has no self–loops and graph is a dag
123   gase/dag[flow]
124 }
```

We import the Graph library which defines classical graph theory constructs such as nodes, graphs, and directed acyclic graphs. First, the abstract signature *sbp* is defined. *Sbp* is a set of relations between two nodes, called *flow*. This defines a ternary relation as a binary relation between an *sbp* and a binary relation *flow*.

By setting constraints over this *flow* relation we obtain the desired structure of *sbps*. The constraint is in the line before last (119) of the previous code snippet, and specifies that the flow relation must comply with the definition of a DAG given in Alloy Graph Library. We copied the relevant definitions from the Graph and relation libraries in the Alloy code snippet underneath.

```
125 module util/graph[node]
126 open util/relation as rel
127 [···]
128 // graph in undirected
129 pred undirected [r: node→node] {
130   symmetric[r]
131 }
132
133 // graph has no self–loops
134 pred noSelfLoops[r: node→node] {
135   irreflexive[r]
136 }
137
138 // graph is a dag
139 pred dag [r: node→node] {
140   acyclic[r, node]
141 }
142 [···]
143 module util/relation
144 [···]
145 // r is acyclic over the set s
146 pred acyclic[r: univ→univ, s: set univ] {
147   all x: s | x !in x·^r
148 }
149 [···]
```

Node Helper Functions

The following three Alloy functions are helpful when implementing our formalization. First, the *nodes()* function returns all the nodes of a given business process. The *preds(Node)* resp. *succs(Node)* give the predecessors and resp. the successors of a Node in the process.

```
150 fun SBP::nodes: set Node{ // set of nodes in business process
151   (this·flow)·Node + Node·(this·flow)
152 }
153 fun SBP::preds[n': Node]: set Node {
154   {n: Node| n→n' in this·flow}
155 }
```

```
156 fun SBP::succs[n: Node]: set Node {
157   {n': Node| n→n' in this·flow}
158 }
```

Operational Nodes

Next to *AseNodes*, a number of other *Node* types are needed in *SBPs*. We call these the operational nodes, because they serve the purpose of defining an execution (routing) workflow along the various *AseNodes*. There are six types of such *Nodes* (see Definition 4.2.1). The *Start* is the node that starts the execution. The *End* node ends the execution.

```
160 abstract sig OpNode extends Node {}
161 sig AndSplit, AndJoin, XorSplit, XorJoin, Start, End extends OpNode{}
```

The other four nodes are control flow rules. We have two types of these. The *Split* nodes open a block in the *SBP*, and the *Join* nodes close a block in the *SBP*.

The operational nodes can also be divided along another criterion: *AND* and *XOR* blocks. These are the two kinds of *LogicBlocks* we have. We need two nodes to define an *AND* block, *AndSplit* opens the block, while a *AndJoin* closes the block. In similar fashion, a *XorSplit* opens a block, and a *XorJoin* closes a block. But we get to blocks a little later in this section in more detail.

A Consistency Constraint on Start and End Nodes

We know that we have a single *Start* node and a single *End* note as well. This is encoded in the Alloy fact *singleStartAndEnd*.

```
162 fact singleStartAndEnd {
163   one Start && one End
164 }
```

Definition of SBP Blocks

Every *Block* contains a sequence of blocks (*blockSeq*) as well as a set of blocks (*blockSet*). Both constructs are partly redundant, since the information in the one relation (*blockSeq*) can be used to build the second relation (*blockSet*). However, we need this redundancy for practical reasons in writing short functions and constraints in the Alloy model.

```
165 abstract sig Block{
166   blockSeq: seq Block,
167   blockSet: set Block // need this for nodeBlocks function (redundant)
168
169 }{
170   lone b: Block| this in b·@blockSet
171   not blockSeq·hasDups // no duplicates
172   blockSet = blockSeq·elems // set reflects sequence
173 }
```

In the Alloy code snippet above, we further constrain the *Block* signature as follows. First, every *Block* can be contained in at most one other block, otherwise its definition makes little sense if any. We write an Alloy function called *hasDups()* to say that the sequence of blocks does not contain any redundant blocks. Similarly to the constraint modeled above, this is to avoid trivial instance models where the same blocks are found many times, which is obviously of little interest to a compliance checking.

We also make sure that the *blockSet* and the *blockSeq* are consistent with regard to each other by saying that the *blockSet* is actually made out of all the elements in the sequence of blocks *blockSeq*.

Node Block as Atomic Block

We define a special type of blocks called *NodeBlocks*. *NodeBlocks* only contain a single (implicit *one* quantifier in Alloy relations) *AseNode*. A *NodeBlock* also has an empty *BlockSeq*.

```
174 sig NodeBlock extends Block { // atomic block
175   node: AseNode
176 }{
177   no blockSeq
178 }
```

SBP Composite Blocks

A *CompositeBlock* inherits from *Block* and contains strictly more than one *Block*. In the following, we will see that we define three kind of *CompositeBlocks*.

```
179 abstract sig CompositeBlock extends Block {
180 } {
181   #blockSeq> 1
182 }
```

Sequence Blocks

The first kind of *CompositeBlocks* are *SequenceBlocks*, which are sequences of *NodeBlocks*. SequenceBlocks do not contain any other *SequenceBlocks*. We dispose of two useful functions, *first()* resp. *last()* which return the first resp. the last *NodeBlock* in the sequence. The technique used is to find the last Block in the sequence of sub-Blocks, and if the last sub-Block is not a NodeBlock, it means the first (resp. last Node) is either an AndSplit or and XorSplit (resp. either an AndJoin or an XorJoin). We obtain the desired result by computing the relational join of two sets, e.g., Block b (the first resp. last sub-block of the Block) and the set of all splits resp. joins.

```
183 sig SeqBlock extends CompositeBlock {
184 }{
185   no blockSet & SeqBlock // wlog sequence blocks do not contain sequence blocks
186 }
187 fun SeqBlock::first: Node {// first node of sequence block
188   let b = this·blockSeq[0]|
189   b in NodeBlock implies b·node else b·split
190 }
191 fun SeqBlock::last: Node {// last node of sequence block
192   let b = this·blockSeq[this·blockSeq·lastIdx]|
193   b in NodeBlock implies b·node else b·join
194 }
```

Control Flow Rules: AND and XOR Blocks

A special kind of *CompositeBlocks* are *LogicBlocks*, which are either *AndBlocks* or *XorBlocks*. Each *LogicBlock* contains a single *Split* node which opens the block and a single *Join* node which closes the block.

```
195 abstract sig LogicBlock extends CompositeBlock{
196   split: OpNode,
197   join: OpNode
198 }
199
200 sig XorBlock extends LogicBlock {
201 }{
202   split in XorSplit
203   join in XorJoin
204 }
205
206 sig AndBlock extends LogicBlock {
207 }{
208   split in AndSplit
209   join in AndJoin
210 }
```

Operational Nodes and Blocks: Consistency Constraints

The *noOrphanOps* fact specifies that for all nodes which are neither a *Start* nor an *End* node, we must have the following properties:

- Every *AseNode* is contained in exactly one *NodeBlock*.
- We have that every *LogicBlock* has one *split* field and one *join* field.
  - For every *AndSplit* resp. every *XorSplit* there is a single *LogicBlock* for which the *AndSplit* resp. *XorSplit* is the *split* field.
  - Symmetrically, for every *AndJoin* resp. every *XorJoin* there is a single *LogicBlock* for which the *AndJoin* resp. *XorJoin* is the *join* field.

```
211 fact noOrphanOps {
212   all n: Node-(Start+ End)|{
213     n in AseNode implies one b: NodeBlock| n in b.node
214     n in AndSplit + XorSplit implies one b: LogicBlock | n in b.split
215     n in AndJoin + XorJoin implies one b: LogicBlock | n in b.join
216   }
217 }
```

The Root Block

There is a single *Root* Block in every *SBP*. The *Root* Block is a *CompositeBlock* which is contained in no other *CompositeBlock*. Another constraint is that all *Blocks* of the *SBP* are contained in the *Root* Block. This containment is calculated by the *getBlocks* function which traverses the *blockSet* relation transitively (using Alloy's reflexive and transitive closure operator) and collects all the blocks reachable through this relation.

```
218 pred Block::isRoot {
219   no b: CompositeBlock| this in b.blockSet
220   all b: Block| b in this.getBlocks// all nodes reachable from root block
221 }
222
223 fun Block::getBlocks: set Block{
224   this.*blockSet
225 }
```

Helper Functions for SBP Blocks

The *numBlocks* returns the number of blocks contained in a *Block* by returning the cardinality of the set *BlockSet*. The functions *firstNode* resp. *lastNode* returns the first node resp. last node in the block. The predicate *hasEdge(Node, Node)* returns true if both nodes by structural induction on the block structure. It distinguishes between the case where we have a *SeqBlock* and the case where we have a *LogicBlock* because we do not have any *OpNodes* (e.g., AndSplit or XorJoin) in SeqBlocks.

```
226 fun Block::numBlocks: Int {
227   #(this·blockSet)
228 }
229 fun Block::firstNode: Node{
230   this in NodeBlock implies this·node
231   else (this in SeqBlock implies this·first else this·split)
232 }
233 fun Block::lastNode: Node{
234   this in NodeBlock implies this·node
235   else (this in SeqBlock implies this·last else this·join)
236 }
237 pred Block::hasEdge[u,v: Node]{
238   (this in SeqBlock
239     && some i: this·blockSeq·inds – this·blockSeq·lastIdx |
240       (this·blockSeq[i]·lastNode = u &&
241       this·blockSeq[add[i,1]]·firstNode = v)
242   )
243   ||
244   (this in LogicBlock
245     && some i:this·blockSeq·inds |
246       (this·split =u && this·blockSeq[i]·firstNode = v)
247       ||
248       (this·join=v && this·blockSeq[i]·lastNode = u)
249   )
250 }
```

Enforcing the Block Structure

This code snippet shows how we further constrain the SBP structure to comply with the block structure we want for SBPs. It does this by structural induction on blocks. It uses the *flow* relation and checks whether for every two nodes we either the following property: these two nodes are linked by an edge (flow) inside the same block, or else, exactly one of both nodes is either the start resp. end node and the other node is the first node of the root block resp. the last node of the root block.

```
251   pred SBP::conformsToBlockStructure{
252   all u,v: Node| (u→v in this·flow iff
253     (some b: Block| b·hasEdge[u,v]
254     or some b: Block | (
255       b·isRoot && (
256         (u in Start && v = b·firstNode) || (v in End && u= b·lastNode)
257         )
258       )
259     )
260   )
261 }
```

Consistency Constraints on Blocks: Global Facts

Obviously, we only want a single *RootBlock*, and set the fact forcing the *SBP* to comply with the block structure defined in predicate *conformsToBlockStructure*.

```
fact singleRootBlock{// a single root block represents the business process
  one b: Block| b.isRoot
}

fact blockConformance{ // the block structure should correctly reflect the business process
  SBP::conformsToBlockStructure
}
```

4.3.1 Examples of Defining SBP Models

In fact, a more accurate word that should be used instead of 'Defining' might be 'Generating'. The reason might be obvious to the reader, since business processes in our approach are declaratively defined. In order to model a business process, it is sufficient to declare the elements (i.e., all kinds of required blocks, since our process language is block-based) in a consistent way with the formal semantics. In the following, we show how a set of elementary and small business processes may be defined.

Code 4.3.1.

```
sig AseNode extends Node {
  //se: ASE
}

fact noOrphanAse {
  all a:ASE | one n: AseNode | n.se = a
}
```

Note that in order to just generate the workflow, which we call SBP model and not the full system model which also includes the ASEs, all we need to do is to comment the lines in the SBP Alloy module which embed ASEs into an SBP model. This is shown in the code snippet 4.3.1. What this causes is that the structural links (Alloy Relations) between AseNodes and ASEs are removed. Thus the Ally model checker does not include instances of ASEs in the generated instances. This decoupling between SBP (workflow) and ASEs has been motivated earlier. It allows us to easily reuse the same workflows with different ASEs to create different system models.

4.3.1.1 Sequence

Conceptual Model

Here, we model the sequence workflow with two tasks shown in Figure 4.9. The code required to generate this sbp workflow is given in the code listing 4.3.2. All that is necessary is to declare two NodeBlocks that correspond to the two workflow tasks and to order these NodeBlocks in a sequence. This is done in the fact signature of the SeqBlock sb1 by setting the mapping of the indexes of the blockSeq property of a SeqBlock (e.g., '$0 -> nb1$' sets the NodeBlock '$nb1$' to be the first in the sequence, etc.). Note that the '@' is required since the statement is inside a signature fact and not in a separate Alloy fact. The following examples follow the same pattern.

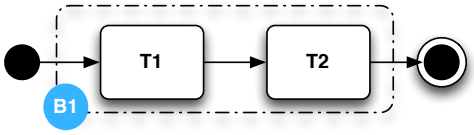


Figure 4.9: Sequence Block

Alloy Model

Code 4.3.2 (Sequence Workflow).

```
276 module Examples/sbp1
277
278 open sbp
279
280 one sig nb1, nb2 extends NodeBlock{}
281
282 one sig sb1 extends SeqBlock{}{
283   this.@blockSeq = 0→nb1+ 1→nb2
284 }
```

Running the Alloy Model

The following Alloy code runs the model described above. Adding the predicate *show()* allows to configure under which conditions the model is to be run. We just need to specify that we want no ASEs to be generated since at this stage we only want valid SBP workflows to be generated. We set a scope of 4 since we know we have at least 4 different nodes: 2 AseNodes, a Start and an End Node.

Code 4.3.3.

```
285 pred show(){
286   no ASE
287 }
288 run show for 4
```

The resulting Alloy instance is shown in Figure 4.10. The figure presents a graphical view on all model instance elements (called atoms) that are shown in yellow coloured squares. The relations (instances) between these atoms are shown as directed vectors. Each vector is typed using a different colour, and the types corresponding to the colours are shown in the block on the upper left part of the diagram (blockSeq, blockSet, flow and node Alloy relations).

Let us start by visualizing the workflow in the model instance. As explained in the Alloy model for SBP, the workflow is encoded through the flow relation of the signature SBP defined between nodes. For instance, the sbp/SBP (the prefix sbp/ tells us the Alloy module where SBP is defined) atom representing the SBP model instance has three outgoing instances of the Flow relation. Each instance of these relations follows this format: $flow[Node1] : SBP \rightarrow Node2$. This is an equivalent notation for this SBP relation: $SBP.flow(Node1, Node2)$, or the following ternary relation $flow(SBP, Node1, Node2)$. The same can be said about the *node* relation represented by the green vectors linking NodeBlocks (e.g., nb1) with AseNodes (e.g., $node(nb1 \rightarrow sbp/AseNode1)$.

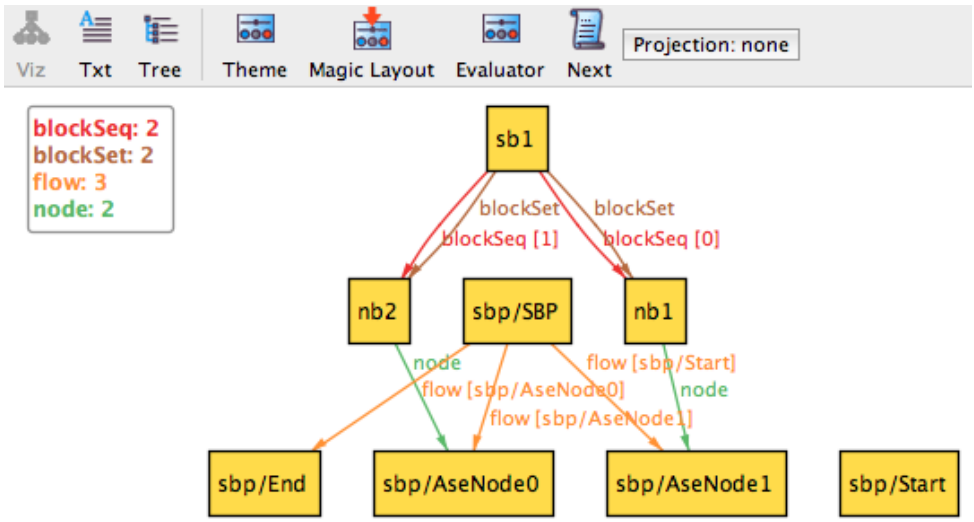


Figure 4.10: Result of Running an Alloy Model - sbp1 Example

4.3.1.2 SeqBlock and XorBlock

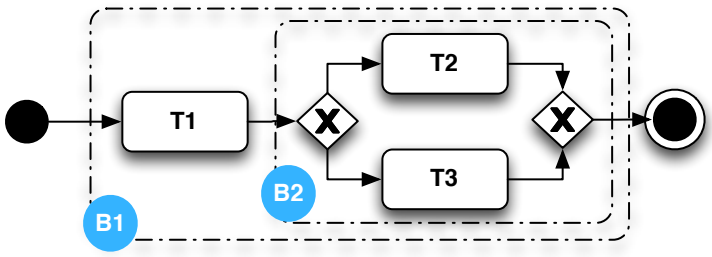


Figure 4.11: Sequence with Xor Block

Similarly to the previous example, we first identify the required blocks. In this case all we need is an XorBlock embedded into a SeqBlock. Then we declare the required NodeBlocks and place them in the right blocks. Here we may notice two things. First the ordering of Blocks (including NodeBlocks) is only important in SeqBlocks.

The other is that setting both the blockSet and blockSeq relations of a block introduces some redundancy. The reason for this is that in the SBP Alloy module, we define blockSet using blockSeq, as a flattened non-ordered set of the elements included in a blockSeq. Consequently, it is a better style to remove the lines in the SeqBlock sb1 and XorBlock xb1 where the blockSet is defined. We kept these in our example for illustration purposes.

The next examples will not define blockSets explicitly, and it is sufficient to define blockSeqs, although the ordering of blocks in a blockSeq is not important for execution in neither AndBlocks nor XorBlocks. For the sake of saving space, we will not show the produced Alloy model instances for the next SBP examples as these grow bigger and more complex.

Code 4.3.4 (Nested Sequence and Xor Workflow)**.**

```
289 module Examples/sbp2
290
291 open sbp
292
293 one sig nb1, nb2, nb3 extends NodeBlock{}
294
295 one sig sb1 extends SeqBlock{}{
296   //blockSet = nb1+ xb1
297   this.@blockSeq = 0→nb1+ 1→xb1
298 }
299
300 one sig xb1 extends XorBlock{}{
301   //blockSet = nb2+ nb3
302   this.@blockSeq = 0→nb2+ 1→nb3
303 }
```

4.3.1.3 AndBlock of SeqBlock

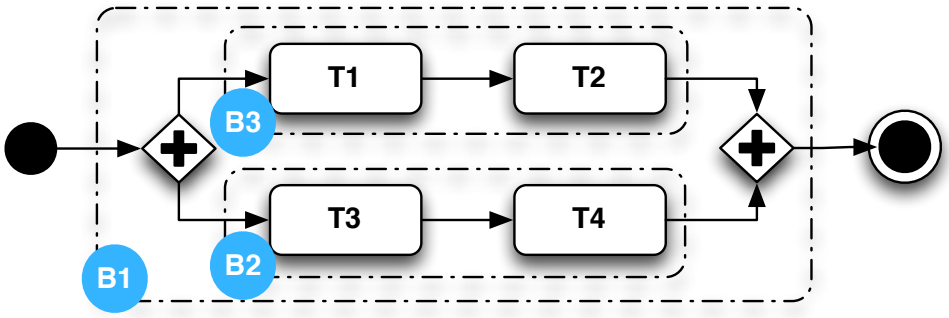


Figure 4.12: Xor Block of Sequences

This example shows how to generate an AndBlock with two branches which contain each a SeqBlock. We need to declare four NodeBlocks. We follow by defining the two distinct SeqBlocks and the containing AndBlock. In Alloy models, the order of declarations is not important.

Code 4.3.5 (Nested Sequence and And Workflow)**.**

```
304 module Examples/sbp3
305
306 open sbp
307
308 one sig nb1, nb2, nb3, nb4 extends NodeBlock{}
309
310 one sig sb1 extends SeqBlock{}{
311   this.@blockSeq = 0→nb1+ 1→nb2
312 }
313
314 one sig sb2 extends SeqBlock{}{
315   this.@blockSeq = 0→nb3+ 1→nb4
316 }
317
318 one sig ab1 extends AndBlock{}{
```

```
    this.@blockSeq = 0→sb1+ 1→sb2
}
```

4.3.1.4 Nested Blocks

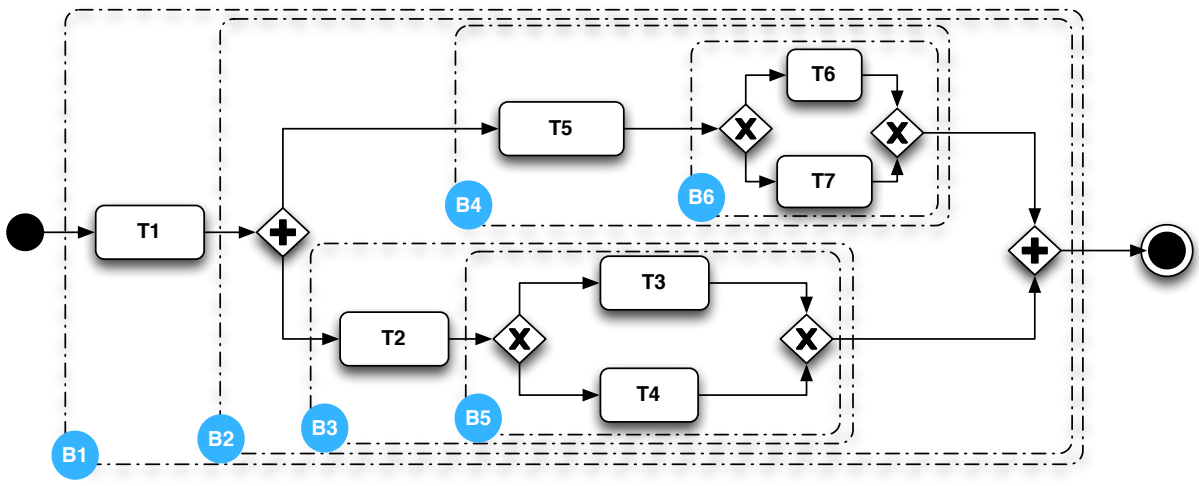


Figure 4.13: Nested Sequence and And Blocks

As a last example, we consider a slightly more complex workflow than the previous elementary ones. The complexity might be defined in various ways, but what is intended here is an informal consideration based on the number of various tasks, the number of blocks, and the intricacy of nesting of these blocks.

The example contains seven different tasks, and 6 distinct blocks: 2 XorBlocks, 1 AndBlock, 3 SeqBlocks. Once the hierarchy of blocks is identified (the process can always be represented as a tree of blocks, equivalent to the directed acyclic graph representation), it is quite straightforward to define the corresponding Alloy model, and an understanding of the previous 3 elementary examples allows the reader to easily interpret the code in the Snippet 4.3.2.

Code 4.3.6 (Nested Blocks).

```
module Examples/sbp4

open sbp

one sig nb1, nb2, nb3, nb4, nb5, nb6, nb7 extends NodeBlock{}

one sig sb1 extends SeqBlock{}{
    this.@blockSeq = 0→nb1+ 1→ab2
}

one sig ab2 extends AndBlock{}{
    this.@blockSeq = 0→sb3+ 1→sb4
}

one sig sb3 extends SeqBlock{}{
    this.@blockSeq = 0→nb2+ 1→xb5
```

```
}

one sig sb4 extends SeqBlock{}{
  this.@blockSeq = 0->nb5+ 1->xb6
}

one sig xb6 extends XorBlock{}{
  this.@blockSeq = 0->nb6+ 1->nb7
}

one sig xb5 extends XorBlock{}{
  this.@blockSeq = 0->nb3+ 1->nb4
}
```

4.3.2 Example of Defining a System Model

In order to define a system model, we need to bind the previous SBP models with ASEs, so that the description of the workflow is extended with the actual ASEs which are being executed. We will show how to do this for the sequence example in 4.3.1.1. The Alloy code in Snippet 4.3.7 shows how to do this.

Code 4.3.7 (Extending SBP with ASEs - 1)**.**

```
one sig ase1, ase2 extends ASE{}

fact{
  (nb1.node).se = ase1
  (nb2.node).se = ase2
}

pred show(){}
run show for 4
```

We change the SBP module back to contain the relations linking a NodeBlock AseNode, itself linked to an ASE, by uncommenting the relation definition $se(AseNode, AseNode)$. We need to extend the Alloy module in 4.3.1.1 with declarations of the two needed ASEs ase1 and ase2. Then we need to define a fact linking each NodeBlock to the corresponding ASE. We first access the AseNode inside a NodeBlock ($nb1.node$), then we set its AseNode to ase1 ($(nb1.node).se = ase1$).

Thanks to the code in 4.3.7 we can now visualize in Figure 4.14 a valid generated Alloy instance of a system model using the show predicate. In this example however, we see the ASEs (e.g., ase2) and two distinct triples of Action, Subject and Entity: (sbp/ase/Action0, sbp/ase/Subject, sbp/ase/Entity) and (sbp/ase/Action1, sbp/ase/Subject, sbp/ase/Entity).

Code 4.3.8 (Extending SBP with ASEs - 2)**.**

```
one sig a1, a2 extends Action{}
one sig e1, e2 extends Entity{}
one sig s1, s2 extends Subject{}
fact{
  ase1.a = a1
  ase1.s = s1
  ase1.e = e1
```

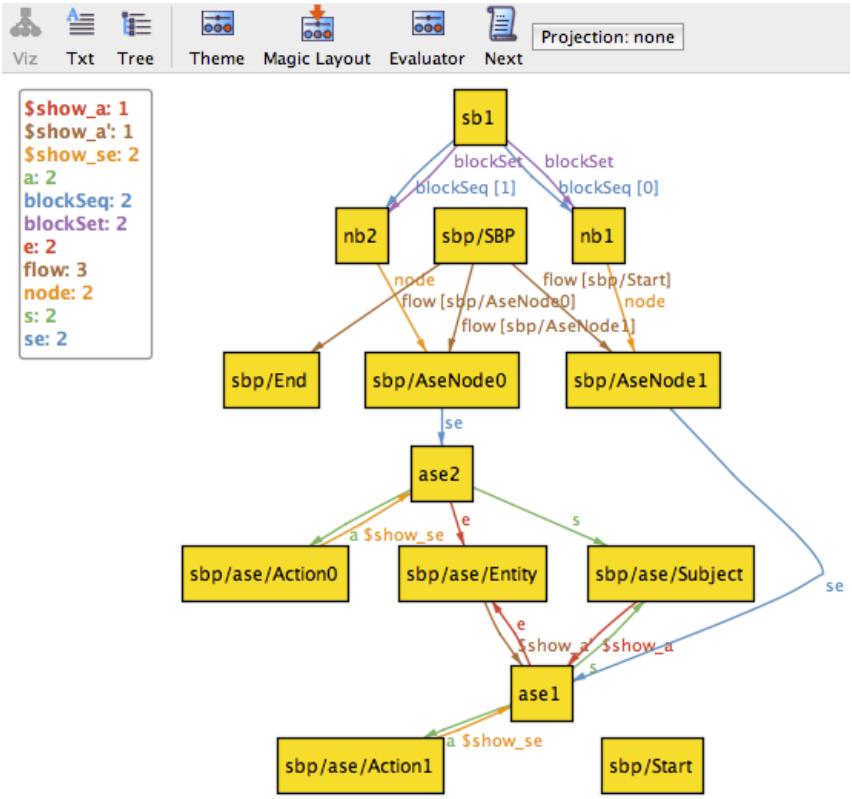


Figure 4.14: Result of Running an Alloy System Model - sbp1 Example

```
    ase2·a = a2
    ase2·s = s2
    ase2·e = e2
}
```

If we want to explicitly specify which actions, subjects, entities make up every ASE, then we need the additional lines of code shown in Code Snippet 4.3.8. This generates the Alloy instance shown in Figure 4.15.

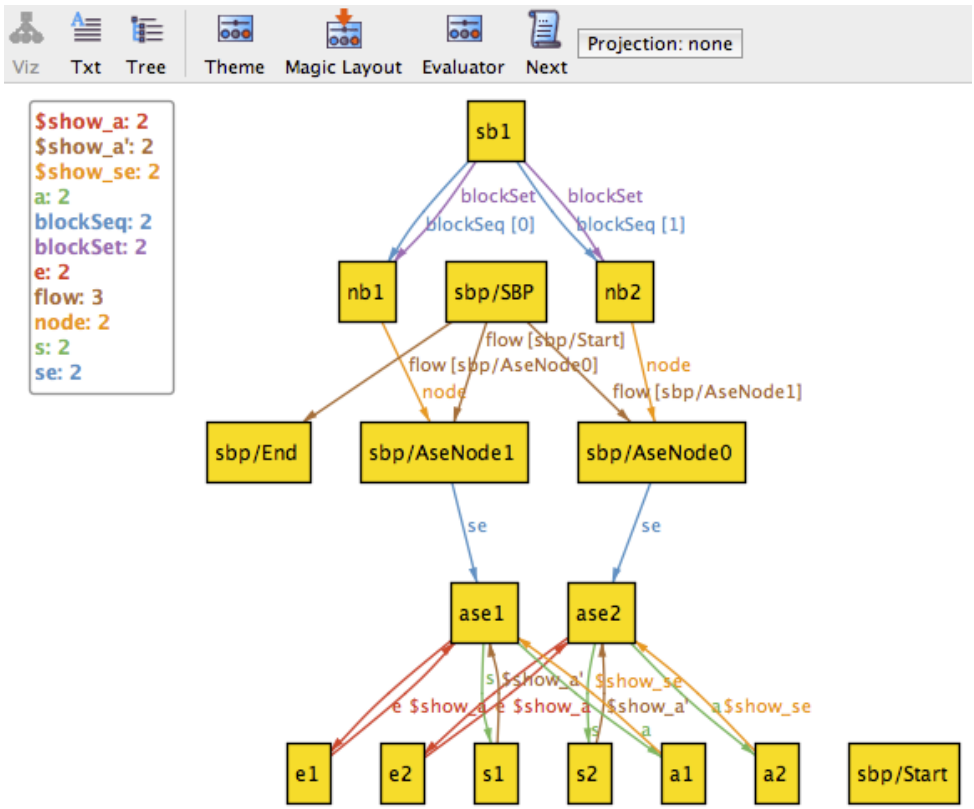


Figure 4.15: Result of Running an Alloy System Model with defined ASEs - sbp1 Example

4.4 Formalizing the System Execution Semantics

Definition of Process State

We start this Alloy module by defining the signature for process states. The signature defines a relation called *nstate* from every node to exactly one state called *ExecState*. As its name suggests, *ExecState* encodes the current execution state of a node in the process.

```
370 module sbp_exec
371 open sbp
372 some sig ProcessState{
373   nstate: Node →one ExecState
374 }
```

Covered Process States

We distinguish between two execution states for a node: it is either *inactive* or *active*, as illustrated in Figure 4.16. As the focus of our work is on the enforcement of policies, our immediate objective is to show how the semantics of policies can be formalized and integrated with formal execution semantics of business processes.

At this stage of research, there is not much added value in our view, in considering complex state models for processes or process tasks. We argue however, that this could show to be a valuable direction of future research. The intuition behind this judgement is that more complex state models for tasks and processes will lead to richer range of policy decisions.

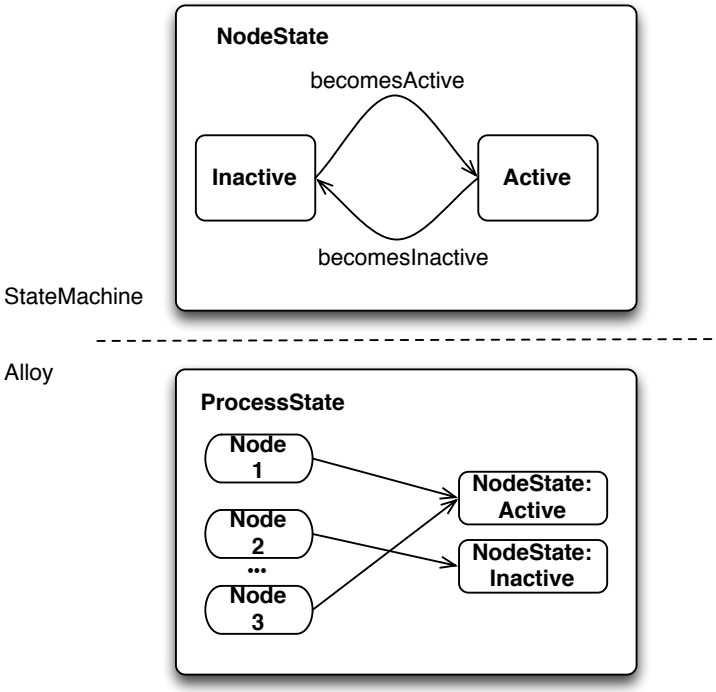


Figure 4.16: ExecState

```
375 abstract sig ExecState{}
376 one sig active, inactive extends ExecState{}
```

Process State Helper Functions

The following functions are defined for the signature *ProcessState* used in the formalization of policy execution semantics: *state(Node)* returns the current *ExecState* of a given Node in the *SBP*. *activeNodes()* returns the set of *Active* nodes in the *SBP*.

```
377 fun ProcessState::state[n:Node]: ExecState{// state of node n in this process state
378   n·(nstate[this])
379 }
380 fun ProcessState::activeNodes: set Node{
381   (this·nstate)·active
382 }
```

Process State Transitions

This part of the Alloy model encodes the state transitions implied by our simple process state model. According to it, we only have two possible state transitions: either an *Inactive* node becomes *Active*, or an *Active* node becomes *Inactive*. In the declarative Alloy model, we encode these two state transitions as Alloy predicates, named *becomesActive* and *becomesInactive*. Both predicates make use of two *ProcessStates*, the current and the next state.

```
383 pred becomesActive[n: Node, s,s': ProcessState]{
384   (s·state[n] in inactive) and (s'·state[n] in active)
385 }
386
387 pred becomesInactive[n: Node, s,s': ProcessState]{
388   (s·state[n] in active) and (s'·state[n] in inactive)
```

```
}
```

SBP Execution - State Transitions

The following predicate is the core of the semantics execution, since it defines the existence condition of a state transition from a process state into the next process state. The basic intuition is that a node can only be activated if the immediate predecessor node in the SBP definition is active. However, we must take the control flow rule routing into account (AND and XOR nodes). We use structural induction on the SBP structure.

```
pred stateStep[bp:SBP, s,s':ProcessState]{
// a node n' can only be active in s' if some predecessor is active in s;
// in that case the predecessor becomes inactive
all n: Node| {
  let succs = SBP·succs[n]|{
    (s·state[n] = active and n in XorSplit)
      implies (becomesInactive[n,s,s'] and one n': succs| s'·state[n'] = active)
    else  (s·state[n] = active and n in AndSplit)
      implies (becomesInactive[n,s,s'] and all n': succs| s'·state[n'] = active)
    else  (s·state[n] = active and no AndSplit & succs)
      implies (becomesInactive[n,s,s'] and one n': succs| s'·state[n'] = active)
    else  (s·state[n] = active and some AndSplit & succs)
      implies (let asplit = succs & AndSplit |
        (all n": SBP·preds[asplit] | s·state[n"] = active)
        implies (s'·state[asplit] = active and all n": SBP·preds[asplit]|becomesInactive[n",s,s']))
  }
}
all n': Node| becomesActive[n',s,s'] implies some n: SBP·preds[n']| s·state[n] = active
}
```

Say the process execution is in a given *ProcessState* s. The predicate implements this decision by first looking at all the successors of any active node and distinguishes between several cases. First, if the current active node in the current state is an *XorSplit* node then exactly one of the immediate successors of the *XorSplit* is chosen non-deterministically and becomes active in the next process state. It is necessary to deactivate the previously active node.

Otherwise, we check if the currently active node is an *AndSplit*. In this case we deactivate the currently active node, then move into a new state where the successors of this state are all active at once. Note that this implementation of the predicate ignores interleaving activations of successor nodes, i.e., two successors are considered to be active at the same time, and the situation where one is activated before the other (i.e., interleaved executions) is ignored. Note that most of the complexity in checking business processes for compliance comes from this interleaving. This hypothesis is designed as a simplification of the compliance problem tackled in this work and allows us to focus our research on the design of a formal approach combining structural and dynamic compliance requirements.

If the currently active node is neither one of those, then we look at whether this node has any AndSplit among its successors. In this case, we deactivate the node, and activate a single one of its successor nodes. Otherwise, in the case where some of the successors of the active node are AndSplits (one or many), we define an intermediary construct called *asplit* which consists of the set of successor nodes of the currently active node which are *AndSplits* and where the predecessor nodes of these *AndSplits* are all active. We then deactivate all the predecessor nodes

of the nodes in *asplit* and activate all the nodes in *asplit*.

One last condition for the predicate to work properly and generate valid execution traces (we will see this in one paragraph further), is that for any node which is activated in the next process state in the trace (s'), we have that at least one of the predecessors of this node in the process which is active in the previous state (s).

Process Execution Trace

Equally important is the predicate which allows to generate valid execution traces of process states. The trace signature carries two fields, one relating it to a sequence of *ProcessStates*, and the other, called *activeSets* relating it to a mapping between an index of Ints and the set of Nodes. This last field is very useful for easily retrieving the information on what node is currently active.

```
409 one sig Trace {
410   states: seq ProcessState,
411   activeSets: Int →Node
412 }{
413   not states·hasDups // no duplicates states
414   states·first in aInitState
415   all i: states·inds − states·lastIdx | stateStep[SBP,states[i], states[add[i,1]]]
416   activeSets·Node = states·inds
417   all i: activeSets·Node | activeSets[i] = states[i]·activeNodes
418 }
```

We define as signature fact the following constraints. We do not allow any duplicate process states, which allows us to generate more full executions traces with a minimum number of process states. The first state on the Trace is a special state called *aInitState* defined further down in this section. Moreover, all the process states ordered in the sequence *states* are linked by the *stateStep(SBP, state, state)* predicate. This predicate simply says that a process state transition exists between every two consecutive process states in the *Trace*.

The set of *Ints* in the domain of the *activeSets* field is exactly the same as the set of indices of the *states* sequence of *ProcessStates*. Finally, the last constraint is that the set of nodes for a given index i in the *activeSets* field is the same as the set of active nodes for a given *ProcessState* in the sequence of states. These two constraints together guarantee that the *states* and *activeSets* fields are always synchronized.

Consistency Constraint on Traces

We do not allow for any free hanging process states, i.e., possible process states which are not contained in the trace. Such generated process states are useless for the verification of properties on the process execution since they are not reachable.

```
419 fact noOrphanStates {
420   all s: ProcessState| s in Trace·states·elems
421 }
```

Initial State in the Process Execution Trace

The process execution trace is initialized with a state called *aInitState*. In this latter state, the only active node is obviously the start node.

```
422 one sig aInitState extends ProcessState{}{
423   // initially start node is active and no other node is active
```

```
424   this·activeNodes = Start
425 }
```

4.4.1 Example of Executing a Simple System Model

We have seen earlier how to model a system model. We reuse the simple sequence example from 4.3.1.1 with a sequence of two tasks to illustrate an execution trace in Alloy. We create a new Alloy module which we name sbp1ex (for execution of sbp1). We then import the Alloy module where the trace-based execution semantics are modelled (SBPExec).

Code 4.4.1 (Snippet)**.**

```
426 module Examples/sbp1ex
427
428 open SBPExec
429
430 one sig nb1, nb2 extends NodeBlock{}
431 one sig ase1, ase2 extends ASE{}
432 one sig sb1 extends SeqBlock{}{
433   this·@blockSeq = 0→nb1+ 1→nb2
434 }
435
436 fact{
437   (nb1·node)·se = ase1
438   (nb2·node)·se = ase2
439 }
440
441 pred show(){
442   #Trace·states > 3
443 }
444 run show for 4
```

Then we declare the same signatures as in 4.3.1.1, without explicit instantiation of ASE triples. We add one more constraint $\#Trace.states > 3$ which forces the number of states to be generated to be at least 4 states. The reason for the number 4 is that we know that executing the SeqBlock will go through 4 states, and we want to force the Alloy engine to generate at least these 4 states so we can have a complete trace.

In oder to show how to execute the example, we will have to use some advanced functionalities of the Alloy tool. We obtain an average sized Alloy model which contains 20 atoms and 44 relation instances. This model can be more easily navigated if we use the projection functionality of the Alloy tool.

Projection on a signature in Alloy allows us to only visualize those atoms which have relations to an atom of that signature. If we project on the signature ProcessState, we may explore the real state of execution of the process by looking into the relations that exist inside each ProcessState. Most importantly, we want to look at the current execution state of every Node in a given ProcessState. This execution state is encoded into the $nstate(ProcessState, Node, ExecState)$ relation.

The following four Figures 4.18, 4.19, 4.20, 4.21 show the four projections in the order of execution that allow us to follow the execution trace of the sequence block example. The sequence is made up of four states in which a single Node is active and is in the following order: Start, First Task (AseNode1), Second Task (AseNode 0), End. The order of the process states in the process execution trace is encoded in the states relation defined in the signature Trace as a sequence of ProcessStates. This is shown using the Alloy tree view on an instance in Figure 4.17.

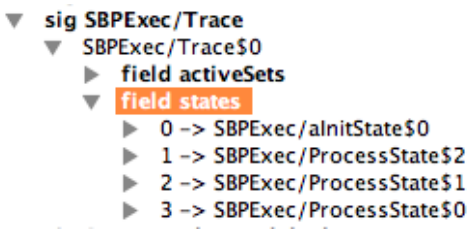


Figure 4.17: Process Execution Trace - Tree View

In order to follow the execution of the system using ordered state, the reader can see the current name of the state in the bottom middle part of each figure. These follow the order shown in Figure 4.17. In each of the four figures, the currently active Node has a relation to the NodeState atom SBPExec/Active (in grey in the figures).

For space reasons, we do not show the execution trace for the other system model examples, as these follow a very similar pattern of modeling. The only difference between the trivial SeqBlock example and the rest is that in the AndBlock we may have states in which several tasks are active at the same time, in different branches of the AndBlock. The XorBlock examples do not allow for this inside the XorBlock itself, unless an AndBlock is nested inside the XorBlock.

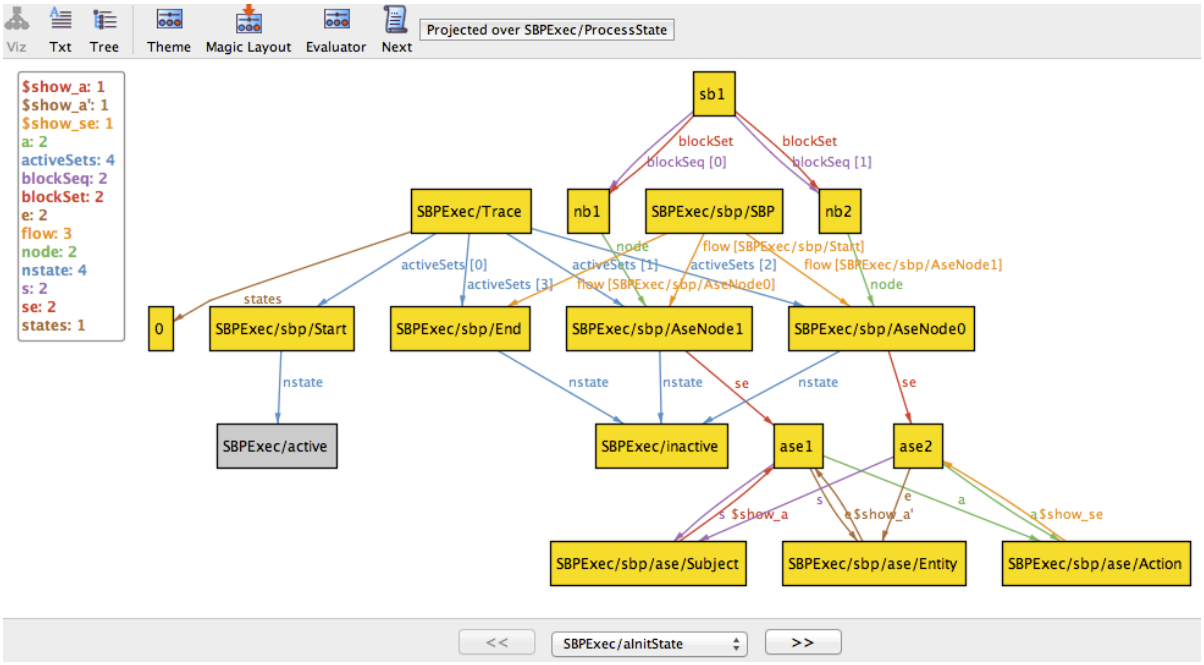


Figure 4.18: Process Execution Trace - Projection on ProcessState - First State

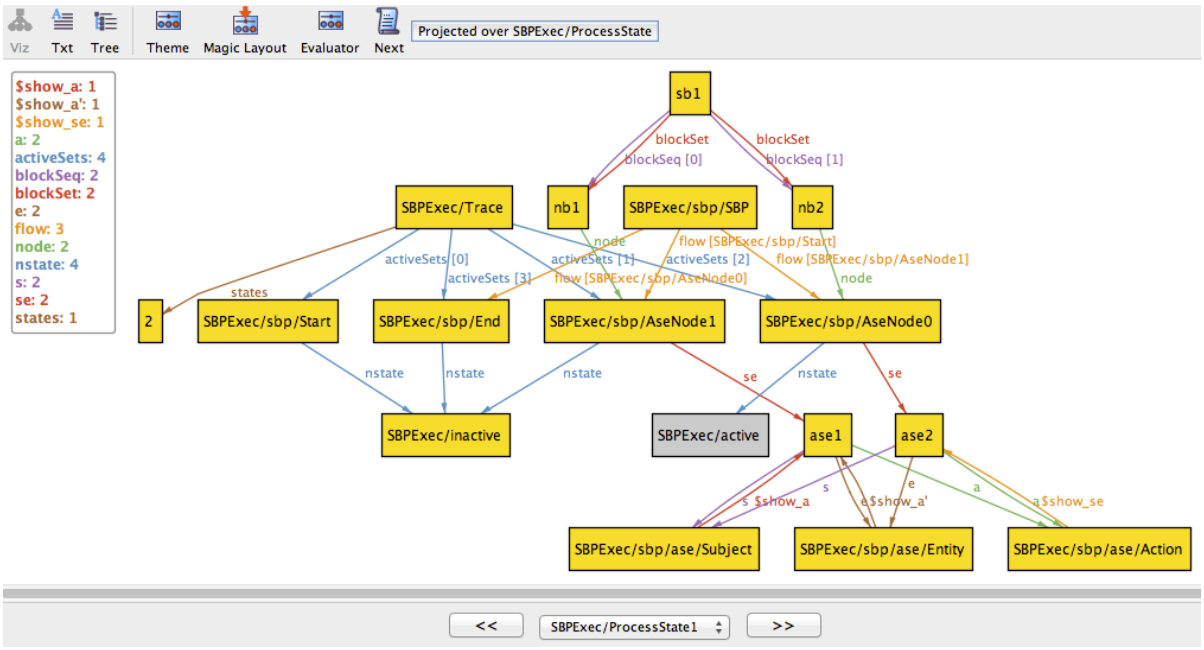


Figure 4.19: Process Execution Trace - Projection on ProcessState - Second State

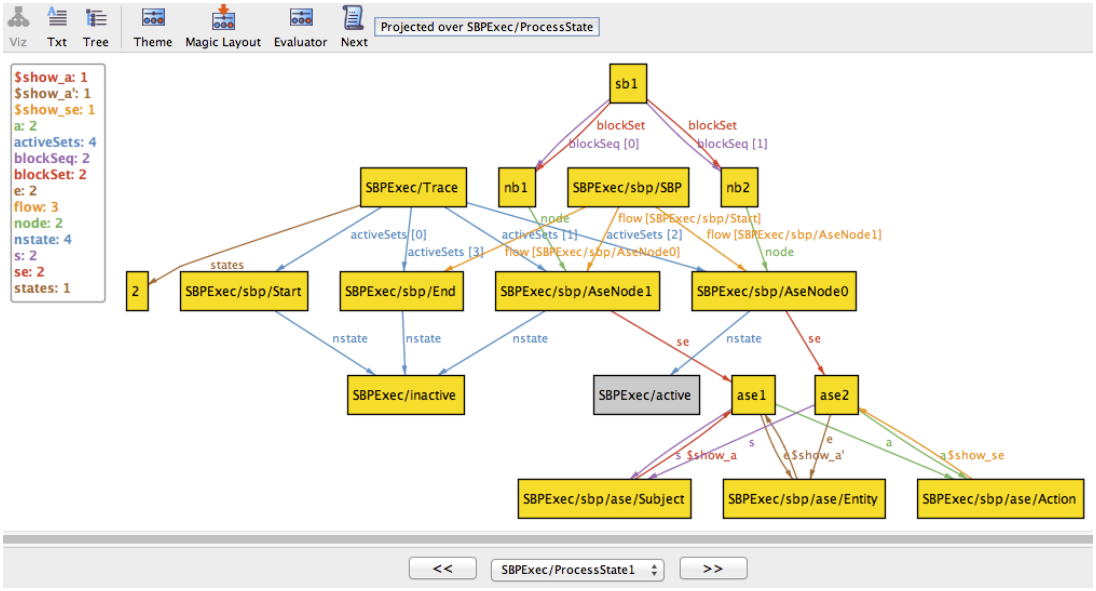


Figure 4.20: Process Execution Trace - Projection on ProcessState - Third State

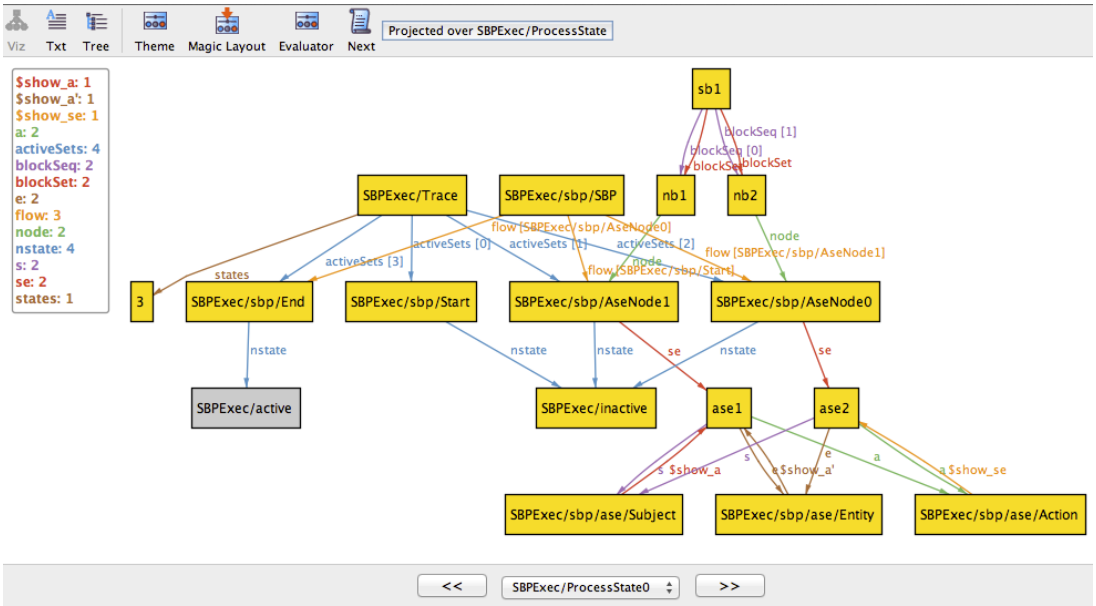


Figure 4.21: Process Execution Trace - Projection on ProcessState - Fourth State

Chapter 5

A Policy-Based Approach to Regulatory Compliance Management

We need a puritanical rejection of the temptations of features and facilities, and a passionate devotion to the principles of purity, simplicity and elegance.
C.A.R. Hoare.

In our approach to a language support to modeling compliance requirements, we rely on the concept of business policy. In CoReL, a business policy is a concept encompassing a decision-making element, as required by a compliance requirement. In this chapter, we introduce the CoReL language for modeling business policies. We start by underlining the motivation and the rationale behind the design of the CoReL language. Following the model-driven engineering paradigm, we then introduce the abstract syntax of the language, and follow by an informal explanation of the semantics of the language. The formal semantics are elaborated on in Chapter 6, while the concrete syntax is presented in Chapter 7.

5.1 CoReL Rationale

The design of the CoReL language is rooted in our study of RCM and of the required capabilities for representing and reasoning on compliance requirements.

5.1.1 Another Look at the RCM Lifecycle

The lifecycle in Figure 5.1 shows the different steps involved in compliance management. These include, among others, the specification (modeling), verification and enforcement of compliance requirements.

Classical expectations on the language will be the ability to break down regulatory texts into related and inter-dependent requirements. Reusability as well and meta-constraints on these regulatory elements is also a concern of high importance, as well as any other strategies used to deal with complexity and increase the flexibility of creation of compliance models using our language.

However, this view of compliance is incomplete, without taking into account the source of the compliance requirements to be managed as well as the systems and models that are impacted by these compliance requirements.

There are many requirements guiding the design of CoReL. We take an agnostic approach to the compliance requirement sources. This implies that CoReL must support multiple regulations (concurrently during a compliance management initiative) and regulation types. Which means

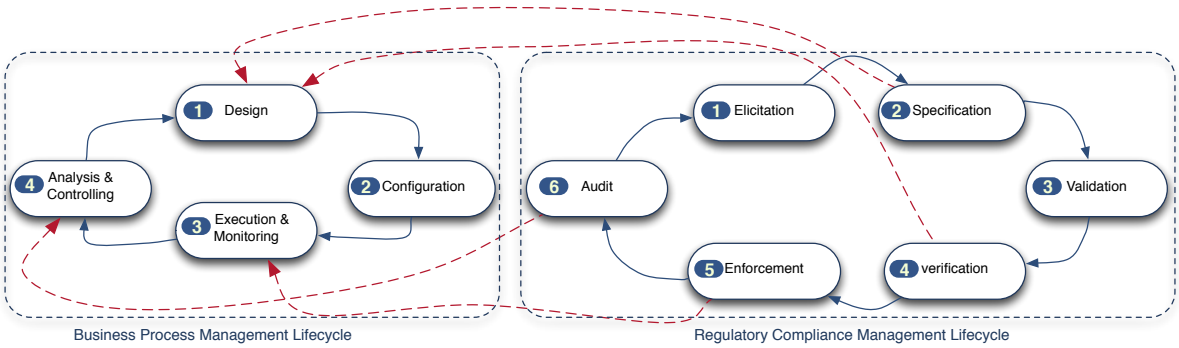


Figure 5.1: Simplified Regulatory Compliance Management Lifecycle- Interaction with a BPM Lifecycle

that we do not represent concepts from a specific regulatory domain which, but wish to stay as independent as possible, and not initiate our research by attempting to model a specific regulation, as we previously did here [KS09].

CoReL must also support multiple types of rules and constraints, as well as multiple business domains. In consequence, we cannot rely on a specific rule modeling language or a logic as a start of the language. Last but not least, the ability to relate regulatory texts to the elements of the (Enterprise) information system they impact must be considered, hence providing traceability between the regulation model and the enterprise (or EIS) model. This is why we decided not to setup our research in the context of a specific business process or enterprise modeling notation or language.

5.1.2 Business Policies as a Semantic Bridge

In the light of the short discussion delivered above, an intuitive strategy vey often used in computer science is to break down and isolate compliance requirements into small, simple, and reusable components. Of course, it is obvious that "small" and "simple" criteria might be interpreted differently and the assessment thereof depends on the usage scenarios. Nevertheless, it is a strategy worth pursuing and evaluating in our work. The final chapters of this dissertation will seek to validate this strategy.

We choose therefore to reify the regulatory compliance requirements by defining a generic modeling concept that will represent bits of the compliance requirements contained in regulations. This concept is what we call a Business Policy. This objectification consists of representing the compliance requirements contained in the regulatory text as separate entities with precise semantics. We propose business policies as a means for this purpose, hence making business policies a semantic bridge between the regulation (i.e. compliance requirements) and the impacted enterprise models.

We also choose to define a set of concepts that will create the interface between the three conceptual domains involved in compliance management, as shown in Figure 5.2: Regulation, Business Domain (i.e. Enterprise Model), and Decision-Making. Section 5.4 will introduce these concepts systematically.

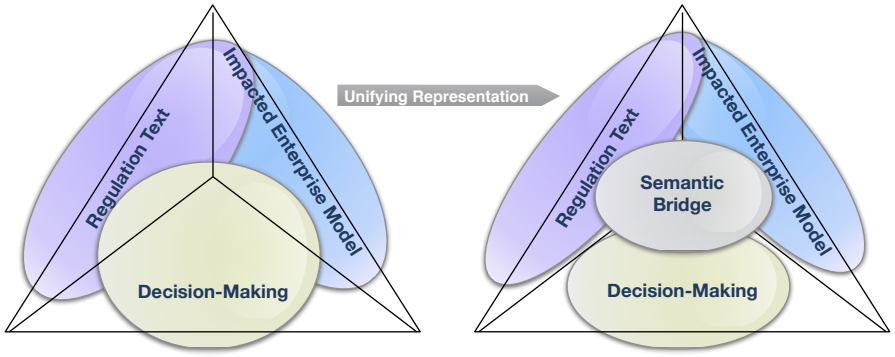


Figure 5.2: 3 Worlds Divide - Regulation, Enterprise and Decision Domains

5.1.3 Language Engineering

5.1.3.1 On Business Policies

A business policy differs from a constraint or any legislation of any kind. While laws for example compel or prohibit behavior (e.g., a law requiring stating some conditions before a building can be constructed), business policies merely guide actions towards desired states (of a system). In more complex cases, business policies are used to align the institution's behavior to a desired behavior. The latter case can be understood as an institution's desire to achieve or avoid a given state of things, or, in a more complex case, come as close as possible to a given sequence of states of things.

To enrich our discussion of business policies, we might refer to the OMG's definition of a business policy. Here, the OMG defines a business policy by distinguishing it from a business rule. The following quoted text is extracted from the SBVR[1] standard document specification [Gro08, Gro13]:

> *Compared to a Business Rule, a Business Policy tends to be less **structured**, less **discrete**, and usually not atomic - that is, not focused on a single aspect of governance or guidance. Also compared to a Business Rule, a Business Policy tends to be less compliant with standard **business vocabulary**, and less **formally articulated**.(...) Business Policies provide broader governance or guidance that is **not directly actionable**.*

It is important to note the subtle difference to more rigid ways of behavioral control. Business policies seek to force the organization to come as close as possible to the desired state of things. Ideally, the organization's behavior is exactly that which is the policy's statement. Our view on business policies is aligned on the distinction from rules given in the SBVR standard. In our work, we propose in the coming chapters a concrete means for decomposing business policies into small and semantically precise building blocks which shall allow policy modelers to combine them in order to define business policies.

Example 5.1.1 (Business Policy I - Purchases Policy)**.**
Many large companies have policies that all purchases above a certain value must be performed through a purchasing process, or by a set of distinct individuals fulfilling a given number of conditions (e.g., second set of eyes policies).

[1] Semantics of Business Vocabulary and Rules

Business policies may be described at various levels of abstraction, some are more concretely implementable than others. The policy in Example 5.1.1 is very general so its implementation will be dependent on the interpretation of the policy modeler or compliance responsible. Business policies use a mix of preferences, interdictions, obligations, requirements on time, location or any other variable of the state of things to express this guidance. If anything, business policies might be best assimilated with guidelines to achieve given business objectives.

Example 5.1.2 (Business Policy II - California Hybrid Cars)**.**
In past years, the numbers of hybrid cars in California has increased dramatically, in part because of policy changes in Federal law that provided 1500$ in tax credits (since phased out) as well as the use of high-occupancy vehicle lanes to hybrid owners (no longer available for new hybrid vehicles). In this case, the organization (state and/or federal government) created an effect (increased ownership and use of hybrid vehicles) through policy (tax breaks, highway lanes).

The definition of a business policy usually follows some intent, such as some desired behavioral change. In the previous example, the intent we may infer is that the government wants to support re-orientation of drivers' choice of cars towards electrical cars. This may be in order to diminish the impact on the environment of car gas emissions. However, ensuring that a business policy indeed leads to such a change in behavior is another, very complex, matter.

Most of the time, policies are instituted to avoid some negative effects (which may have been observed in the institution) or to seek some positive benefit. Not accessorily, organizations also use business policies to describe how the organization should react to deviations from the desired behavior or state of things. Before we define a business policy, we must first define a more fundamental concept, *decision-making*:

Definition 5.1.1 (Decision-Making)**.**
*By decision-making we mean whether to {*allow, force, prohibit, dispense (from)*} a given* Subject *to undertake an* Action *on a given (organizational)* Entity*.*

In this dissertation, we will use the following definition of a business policy:

Definition 5.1.2 (Business Policy)**.**
A business policy is the representation of a regulatory compliance requirement in the form of an enforceable guideline to decision-making.

In Definition 5.1.2, we use the term 'enactable'. By that we mean something which can be represented in a form that makes it operationally usable, i.e., executable, or enforceable, etc, either automatically or by a given instrument such as a human or an institution. Let us take the following example. In China in 2013, the country's government decided to increase taxes on real estate possessions of people possessing two homes or more [2]. This was one amongst many other measures to fight the inflation in real estate prices in the county. However, this led to a dramatic increase of divorces all over the country. The reason was that people used a tiny inconsistency in the law, allowing divorced couples possessing two homes to sell one of these two without paying taxes. The money thus spared can reach the tens of thousands of euros. The people were openly saying they wanted to divorce, sell their real estate, and remarry afterwards. Here the cause for trouble was at least a combination of two factors: (i) a somewhat uncanny dependency between different policies (which possibly happens more often than one would think) as well as (ii) an unconsidered reaction of the policy objects when trying to enforce a given business policy. The

[2] http://bit.ly/WJY5f1. In French. Retrieved on the 7th of March 2013.

point here is that in the absolutely general context, managing business policies is certainly a challenge.

Wrap-Up The previous discussion leads us to assume that here is a semantic divide between regulations on the one side and rules or any logical and structured representations (e.g. rules, constraints) of decision-making on the other side. Moreover, business policies naturally exist at this boundary between the two worlds, but suffer from a number of shortcomings making them useless for automated compliance management. In this scenario, business policies shall represent the atomic decision-making elements implementing a regulation, and business rules shall be the formal means for representing business policies by grounding them in some form of logic. The objective of CoReL is to provide a formal and engineered language for business policies.

5.2
A Deeper Look at Compliance Requirements

5.2.1 The Semantic Gap in Existing RCM Solutions

In Figure 5.3, we show that classical approaches proceed so that constraints are extracted directly from regulations and formally modeled in logical formalisms (i.e. compliance requirements are directly translated into formal constraints). The wide semantic gap between the regulation domain and the formalism domain, raises at least two issues. First, the abrupt jump from regulation documents to constraints risks loss of information that mere constraint specification fails to express. Examples of such information include the applicability conditions of the constraints, as well as the consequence of violating the constraints. Second, Business Users (BUs) who in fact play a key role in the context of RCM modeling, are hardly able to work with formal languages.

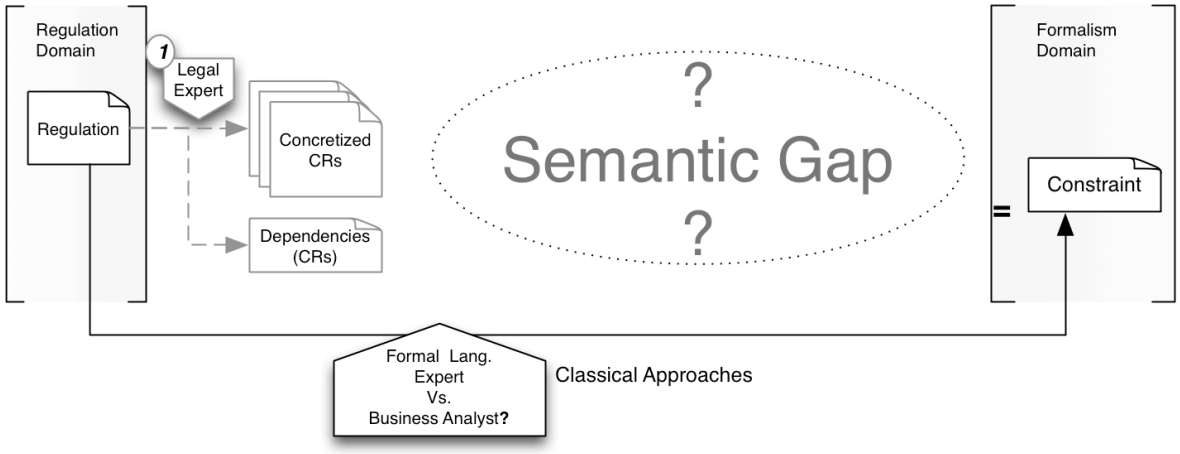


Figure 5.3: Semantic Gap Between Compliance and Logic Modeling

In CoReL another approach is followed. Regulation documents are first 'concretized', i.e., interpreted by legal experts and formulated into a set of guidelines called Compliance Requirements (CRs). Concretized CRs are not a proper concept to CoReL and are widely

mentioned in literature, sometimes referred to as 'internalized' CRs. Such concretization is necessary as regulations are kept abstract on purpose, in order to allow wide adoption by a variety of bodies. Each concretized CR contains (i) the impacted EM elements, (ii) the constraint placed on these elements, (iii) the associated violations and eventually, (iv) the required recovery mechanisms in case of violation of these constraints. Then every CR is modeled as a CoReL policy which is further implemented by CoReL rules. Roughly, CoReL rules represent the constraints classical approaches produce. A very similar methodology is introduced in [SGN07]. However, we claim that our approach seeks to encompass compliance requirements into decision-making elements which can be combined, and directly enforced on executing business processes. On the other hand, the work in [SGN07] rather seeks to organize the compliance engineering work methodologically and create a link to the strategic aspects of managing compliance.

We do not view RCM as a binary problem, i.e., CoReL supports associating several violations to the same CR. Concretely, a company may have to accept that a given violation of a CR happens in case it estimates that preventing this violation will cost too much or hinder a growth opportunity for the company. But it may in this case define a way to recover from the violation. This is an example of the aspects of compliance management which rather pertain to risk management.

Also, a measure of compliance is necessary to allow BUs to judge situations a company finds itself in. Such measures can be defined generally for BP models [LSG08]. Furthermore, as every company is different, a custom quantification of compliance is desirable as part of the reporting component of a RCM solution. The violation models used in CoReL allows to define customized compliance quantifications.

5.2.2 Coping with Regulation Complexity

For the sake of usability, an RCM solution should also provide features to tackle the inherent complexity of regulatory compliance modeling which mainly comes from two sources: (i) logical formalisms used to express constraints are too complex to be used directly by Business Users (BUs), (ii) the number of CRs one has to handle in RCM is usually overwhelming.

To handle this complexity, CoReL modularizes policies to enable easy reuse. A CoReL policy consists of the following parts: (i) an ASE triple, which represents the impacted Enterprise Model (EM) elements, (ii) a context, which models the applicability condition of a policy, (iii) a control, which models the constraint which is enforced by the policy, and (iv) a set of violations. For each violation, an optional recovery can also be associated. All these constructs are defined as building blocks for creating policies. We will see in Section 5.4 how the various reusable building blocks of a policy definition are defined and used. In addition, CoReL provides a graphical notation and makes use of a repository to enable reuse of police building blocks.

5.2.3 2D Classification of CRs

Widely studying literature around the topic of RCM, we observe different types of CRs are tackled using different approaches. We collect and classify the categories of CRs encountered in literature along two dimensions: the type of logical formalism used and the artifacts of the enterprise model considered.

Definition 5.2.1 (CR Categories - Logical Formalism Dimension)**.** *We distinguish three types of CRs along this dimension: structural, temporal and contractual, defined as follows:*

1. Structural CRs (SCRs) *are constraints which hold over the static (structural) part of an EM. For example, a certain document's size must not exceed a certain number of pages. OCL is an example of language used to model these CRs.*

2. Temporal CRs (TCRs) *express temporal dependencies between execution states of a BP. For example, if a customer deposits cash on his bank account, then this amount should eventually show up on his account balance. CTL and LTL are examples of formalisms used to model these CRs. Some formalisms extend classical temporal logics with real-time, such as MTL, to be able to express timing constraints.*

3. Contractual CRs (CCRs) *express duties, rights and commitments that EM elements hold over each other. The obligation to pay a fee every month for a user of an online DVD rental service is an example. Contractual CRs are typically found in contracts and usually require modal (e.g., deontic, alethic) logics to be modeled, such as FCL [GM05] and SBVR [Gro13]. More complexity is introduced when handling mechanisms for violation exceptions are introduced, which implies non-monotonicity. Some CCR languages allow to define valuations CCRs and their violations.* □

Other works have acknowledged this difference and compared languages belonging to the three types of logical formalisms. In [ETvdHP10b, ETvdHP10a], the expressiveness of LTL[3] and CTL[4] for temporal CRs and of FCL[5] (a deontic logic-based language) for contractual ones are compared. SCRs can be expressed in temporal logics using the AG operator (Always Globally) of CTL or the OB operator (Obligation) of deontic logic. TCRs extend SCRs with temporal operators while CCRs extend SCRs with deontic modalities. Languages which can be used for CCRs are SBVR [Gro08] or FCL [GM05]. Languages which can be used for TCRs are CTL, LTL, CTL* [CK08] or μ-calculus. Finally, another layer of complexity is added when structural, temporal and contractual aspects of compliance requirements are combined, as in this example which mixes all three types into one statement: all purchase orders must be verified by an auditor after they have been authorised by a purchaser officer, and before the purchase order is acted upon.

Furthermore, another challenge in RCM is the fact that grasping a company through its EM requires taking several business perspectives into consideration, such as the organizational structure, the usage of resources, the management of goals and objectives, quality, security, risks, etc. We refer to these aspects as the Enterprise Business Aspects (EBAs). A framework for RCM would ideally act as a prism for regulation along the EBAs. Unfortunately, most approaches for CR modeling either stay at a formal level, i.e., no EBA is considered in particular, or at best tackle one EBA, sometimes partially [ACPP11, KL08, SM02, OZD08, PHM+09, KKPP10].

Definition 5.2.2 (CR Categories - EBA Dimension)**.** *We identify three types of CRs for illustration purposes along this dimension: informational, resource usage and organizational, defined as follows:*

[3]Linear-time Temporal Logic
[4]Computation Tree Logic
[5]Formal Contract Language

1. Informational CRs *target the attributes describing an EM element. For example, the size of a document that is transfered between process tasks.*

2. Resource Usage CRs *express constraints that must hold before, during and after using a resource. Examples of resources are web services, persons, databases, etc. Examples of usage are allocating, sending, printing, accessing, etc.*

3. Organizational CRs *express constraints on the organizational elements such as roles carrying out BP tasks, or departments where the processing of a BP task is located. The well-known segregation of duty (SoD) problem is an example.* □

5.2.4 Coverage of 2D CR Space

One aim of research should therefore be the capbility to cover the two dimensions of CR modeling. The following Table 5.1 gives a broad qualitative evaluation of the maturity of existing approaches to RCM projected over the two dimensions elicited above. We see that structural CR modeling is the most supported by research, while works on contractual CRs modeling seem to concentrate on the informational type. Generally speaking, RCM solutions would gain at being extended along the EBA dimension. Moreover, we can observe that merging the expressive capacities of logical formalisms is a desirable feature. An example of merging structural and temporal CR languages is the research stream seeking to combine OCL and CTL as in [MO07].

Table 5.1: Distribution of Approaches Along the 2 Dimensions

	Structural	Temporal	Contractual
Informational	● IS	◐ IT	◐ IC
Resource	◐ RS	◐ RT	○ RC
Organizational	◐ OS	◐ OT	○ OC
Legend: Degree of Satisfaction			
	●Excellent	◐Good	○Unsatisfactory

5.3 A Running Example: Printer Usage

5.3.1 The Example Process

The BPMN [OMG11] model in Figure 5.4 shows a simple process defining the usage of a university printer by university users. The first pool is called "User" and shows a sequence of tasks that the user needs to execute in order to print a given file. The second pool is called "Printer" and shows the tasks that the printer will execute in order to fulfill the user's goal of printing a given file.

The file to be printed is transmitted between the two process actors (Subjects in ASE terms), as represented by BPMN pools using messages[6]. The task *Connect To Printer* sends a message to the printer with a connection request and the printer replies with a message acknowledging the connection. In the following step, the task *Send File* sends the file to be printed to the

[6]BPMN messages are represented as dashed arrows with triangular empty heads that cross pools.

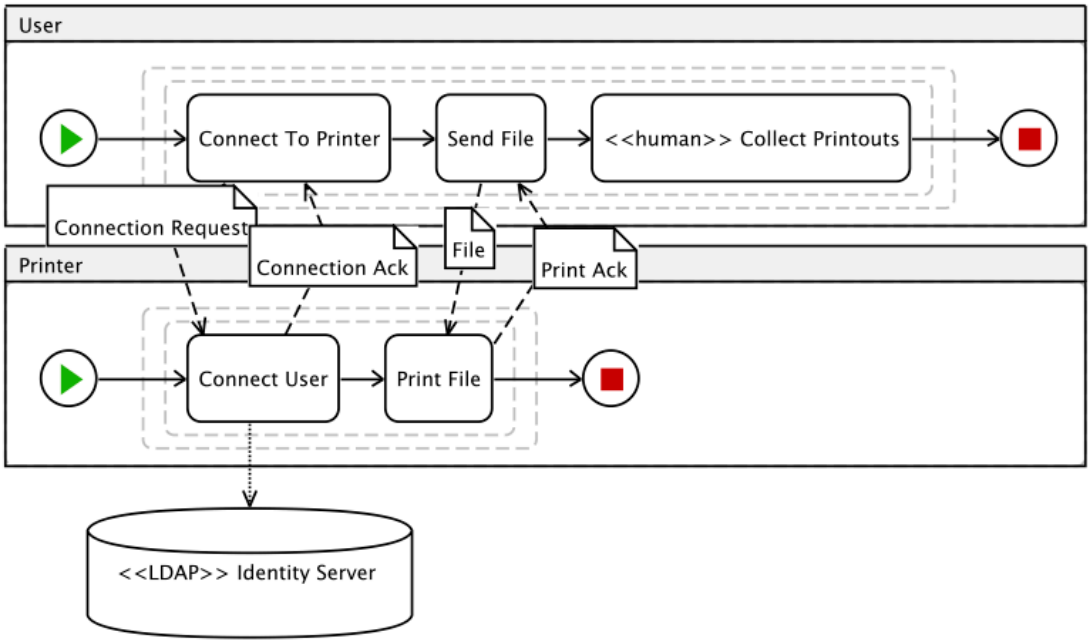


Figure 5.4: The printer Example

printer, and the printer replies with a confirmation that the file was indeed fully printed.

The reader who is familiar with established process modeling notations such as BPMN will recognize many similarities in the concrete syntax used here. The reduced set of modeling constructs can be found across several notations. The pools (horizontal containers containing a workflow) are a standard construct in BPMN and used in many tools or modeling eEPCs [Sch00b]. The shapes looking like files (e.g., *Print Ack*) are so-called artifacts in BPMN and are used to represent resources processed and messages sent in the process. The cylinder shape is called a Datastore and is used to represent data persistence which can be accessed by tasks.

5.3.2 Compliance Rules

In the running example shown in Figure 5.4, we have two concurring business process pools. One models the process executed by a printer, and the other represents the behavior of a printer user. The system administrator, after reading the regulation, defines a list of such CRs based on his knowledge and experience. Table 5.2 lists a purely illustrative set of such CRs, in order to give the reader an impression of the range of constraints we wish to be able to model.

In order to model the previous CRs, we will show in the rest of this chapter how to use the CoReL language. We introduce the syntax of the language and give an informal account of its semantics in the following sections. In the last section of this chapter, we will come back to the process example introduced here and show how a set of CoReL models can be created to represent a given set of CRs.

Table 5.2: Compliance Requirement Examples

CR	Definition
CR1	External students cannot print on university printers. Regular and exchange students can.
CR2	Students are prohibited from printing on the week-end. Faculty members can.
CR3	Students are limited to 400 pages a month, with a bonus of 50 pages if it is their first month at university. Every student who prints less that 30% of his allocated print pages over a whole semester can receive a 5 euro waiver ticket to use the university's swimming pool.
CR4	Faculty members can print up to 2000 pages a month. Faculty members are allowed a violation of the max number of pages if it amounts to less than 10%, in which case they are warned per email. In case they exceed the number of allocated pages by more than 10% pages, their access to printers is stopped.
CR5	All these policies are active over the academic calendar semester for non-faculty members, they are not allowed to print outside of these months.
CR6	If it is discovered that a non-faculty member has accessed printers from outside university computing centers, her account is blocked. If the intruder has no account on the university's systems, then the incident must be logged and a notification email is sent to the system administrator.
CR7	No rewards for low printing faculty.

5.4 Abstract Syntax

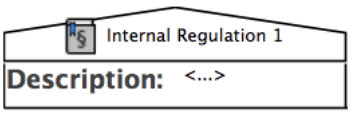
5.4.1 Abstract Syntax

This section introduces the abstract syntax of the CoReL modeling language. An abstract syntax consists of an Abstract Syntax Metamodel (ASM) and associated constraints. The abstract syntax metamodel of CoReL is given in Figure 5.5. For the sake of understability, we group the language elements into four *blocks* shown in Figure 5.5 as dashed boxes. We structure the introduction of the ASMs concepts along these blocks.

The *Policy Block* represents concepts related to the elements of the enterprise model for which the policy is defined, the *Decision Block* includes concepts that model how the compliance check is conducted, the *Recovery Block* models how the compliance decision acts on the enterprise model. The following paragraphs concisely introduce each element of the three blocks. The core concept in the metamodel is the *Policy*. We further refine the definition of *Policy* in the following definition:

5.4.1.1 Policy Block

Regulation



A *Regulation* is used to represent the origin (the regulatory text) of the CR modeled by the *Policy*.

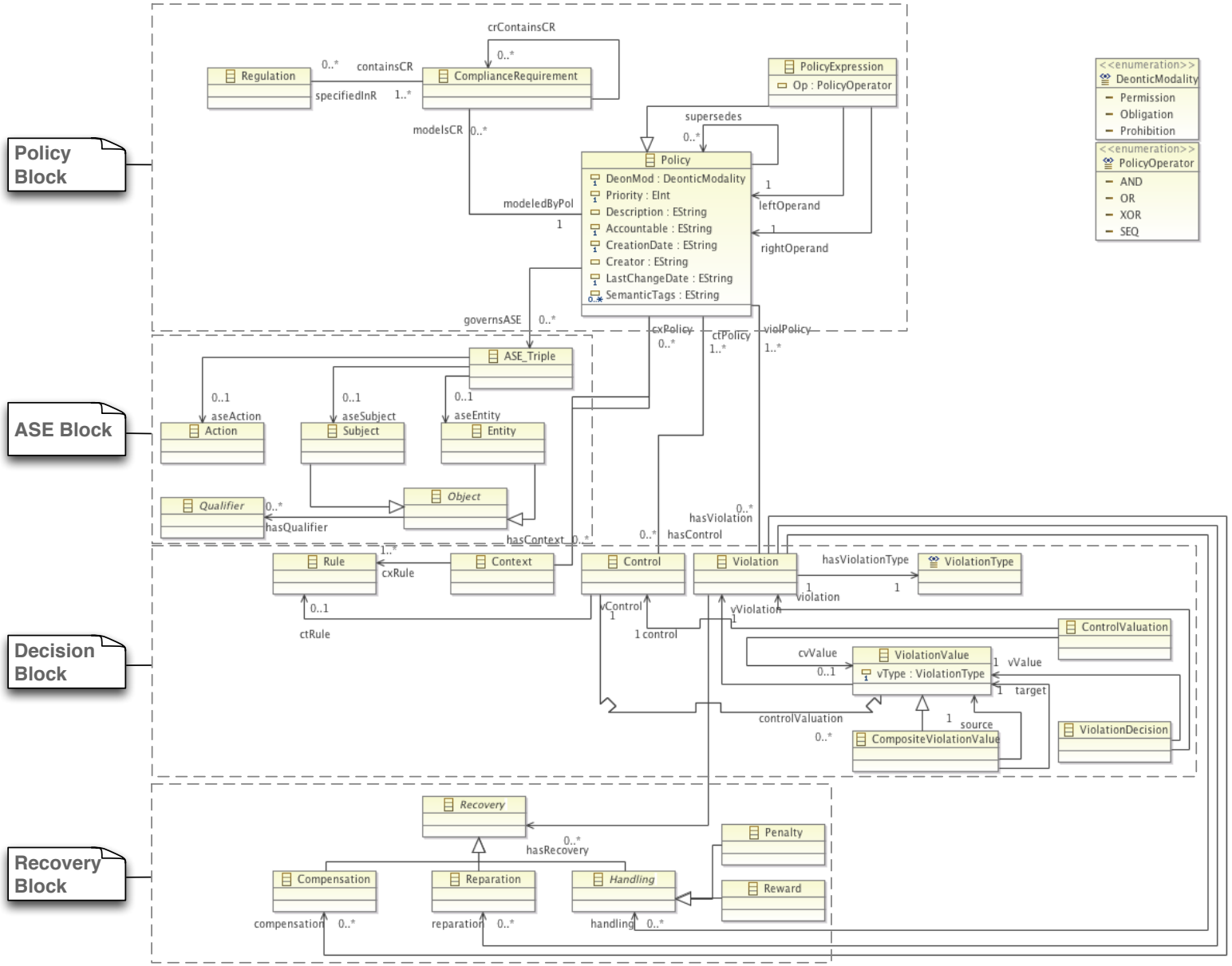
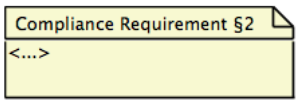


Figure 5.5: CoReL Metamodel [in ECORE]

Compliance Requirement



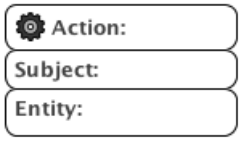
A *Compliance Requirement* (CR) is the textual sub-element of a regulation chosen by the policy modeler. A CR may be of various granularities. Typically, a CR is a paragraph of a law, or of a contract. Selecting a CR from the regulation requires expert knowledge of the regulation as well as the expected complexity of the modeling effort of this CR. This is why this step is best realized by a CoReL and a regulation expert together. Every CR is modeled formally as one CoReL Policy.

Policy

A *Policy* has a deontic modality (Permission, Prohibition, Obligation). A one-to-one relationship links a CR to a *Policy*. A *Regulation* represents the origin of the CR modeled by the *Policy*. A CR in always linked to a single *Policy*. Various metadata attributes are associated with the policy and carry information such as the creator identity, the policy manager, date of creation, date of first instantiation, any specific risks it mitigates or internal controls it implements, etc.

5.4.1.2 ASE Block

ASE Triple



We call $(Action, Subject, Entity)$ an *ASE-Triple* [MEKEP11]. The definition of an ASE-Triple maps *Action* to an enterprise model action, which in our approach applied to the business process domain, is a business process task/activity, and maps both *Subject* and *Entity* to any element of the enterprise model capable of executing the Action, resp. on which the Action is executed. Subject and/or Entity can possibly be defined as empty (*nil*). Definitions similar to our *ASE-Triple* concept exist in research around policy languages [Kag04] but not in literature around enterprise regulatory compliance management (i.e., compliance defined for enterprise models).

Definition 5.4.1 (Action)**.** *Every element of the system is either an object or an action. An action can be implemented by any subject on any entity.*

Definition 5.4.2 (Subject)**.** *Every object can carry the role of a subject if it is capable of executing an action.*

Definition 5.4.3 (Entity)**.** *Every object on which an action can be taken is therefore carrying the role of an entity.*

Subject and Entity are subtypes of the abstract class *Object*. An *Object* has a set of *Qualifiers*. The latter is the concept used to model properties specific to an *Object*. It is possible to express that a given *Subject* has a business role by mapping one of the *Subject's Qualifiers* to the business modeling language specific concept of *Role*.

For instance, we might assume that the business process modeling notation for which we model CoReL *Policies* supports business roles for *Subjects*. Or else, let us assume a metamodel extension of CoReL that defines a business model. An example is given in Figure 5.6. In this

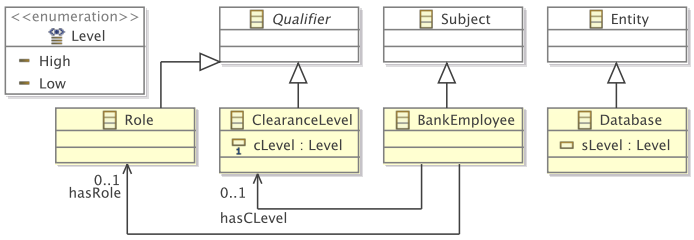


Figure 5.6: Example of a Business Model Extension of ASE

exemplary metamodel it is sufficient to define a relation (i.e., an Ecore eReference in the Figure) between the *Employee* and a *Role*, and another relation from Employee to a class *ClearanceLevel*.

We thus incept the semantics of an ASE triple as follows:

Semantics of an ASE Triple Be the predicate: $Action\ (Subject, Entity)$. We define an ASE triple as: $(ASE) \triangleq [A\ (S, E)]$. In plain English, we say that *Subject* S takes *Action* A on *Entity* E.

5.4.1.3 Decision Block

The decision block encapsulates the decision-making elements of the CoReL language. First, we define the concept of *Rule* in CoReL, followed by those of *Context*, *Control* and *Violation*, *ViolationDecision*, *ViolationValue* and *ViolationTypes*.

Rule

The concepts of Context and Control (see below) build upon the *Rule* concept. Concretely, a Context and Control can only be evaluated if both the ASE triple and the Rule statement that refers to Action, Subject and Entity are provided. The Rule formula must be evaluated. Otherwise, the policy cannot be interpreted and therefore is not enforced.

Definition 5.4.4 (CoReL Rule)**.** *A rule in CoReL is a logical statement written in a rule language. The variables in this statement are enterprise model objects as well as actions (elements). A variety of rule languages may be used as CoReL rules. For instance, in many rule languages, the general form of rules is: Antecedent → Consequence, where Antecedent and Consequence are logical propositions.*

If the antecedent part is set to $True$, then the rule becomes a simple constraint. Any rule language that expresses rules following this format is adequate for use within CoReL. In [PAN04], the reader will find a comprehensive overview of the logics and languages used to define rules. In many of the examples used throughout this thesis, the language we will use is first order logic to express rules.

A Rule in CoReL may be written in various formalisms (LTL, CTL, FCL, OCL, etc.) and is denoted by: $R^{<Name>}_{<Formalism>}$, where $< Name >$ is the name or ID of the Rule and $< Formalism > \in$ {'LTL', 'CTL', 'FCL', 'OCL', ...} tells the formalism used to express the Rule.

Example 5.4.1 (Propositional Rule)**.**
This rule checks whether a variable Date is the current day and is a week-end day, and whether

the user is a student. User is defined as a sub-type (set inclusion semantics) of Subject.
Given predicates: $Weekend(Date), Today(Date), Student(Subject)$.
$R^1_{Prop} = Weekend(Date) \wedge Today(Date) \wedge Student(User)$

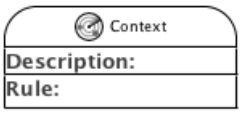
Example 5.4.2 (OCL [Obj10] Rule)**.**
We express the constraint on the Student class that each of its instances be limited to a total of 500 pages to print.

context *Student*
inv *max_spare_pages : self.pages_to_print < 501*

Example 5.4.3 (Temporal (here CTL [BK08]) Rule)**.**
This rule states that the ConnectToPrinter *action must always be eventually followed by the execution of the* Print *of the* Reject *Actions (or both) at least once.*
$R^2_{CTL} = \boldsymbol{AG}[ConnectToPrinter \rightarrow \boldsymbol{EF}(Print\ or\ Reject)]$

Example 5.4.4 (Contractual (FCL [GM05]) Rule)**.**
This rule states that every student who prints a file has the obligation to have enough credit to print all pages of the file. Otherwise, if the student still prints this file, he will have to pay for the number of pages of the file separately.
$R^3_{FCL} =$ *Print(student, file)* $\rightarrow \boldsymbol{OB}_{student}$ *Credit(file)* $\otimes \boldsymbol{OB}_{student}$ *Pay(file).*

Context



The *Context* of a policy formally models the state of the system in which or the set of elements on which the policy is applicable. A *Context* is implemented in our language as a *Rule*.

The context describes under which conditions a policy becomes active, i.e., under which conditions it needs to be enforced. Contexts are used to further refine the constraints that must hold so that the associated policy can be enforced.

For instance, imagine that we want to express a CR on printer users who are students, and we would like this CR to be only applicable on the weekends. For the ASE-Triple defined as $(Action, Subject, Entity) \mapsto (SendFile, User, File)$, we model a Policy P, with context C_x^{Policy} defined as:

Example 5.4.5 (Context)**.** *Given predicates:*
$Weekend(Date), Today(Date), Student(Subject)$
We define the Context named C_x^{Policy} *as:*
$R^1_{Prop} = Weekend(Date) \wedge Today(Date) \wedge Student(User)$
$C_x^{Policy} = R^1_{Prop}$

Control

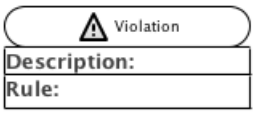
The *Control* specifies the constraints that must hold on the enterprise model. A *Control* is no different from a *Context* in that it is expressed as a *Rule*.

The following example expresses in the control that faculty members cannot print more than 2000 pages. The context states that the printer used must be in the realm of one of the university laboratories, as we do not want the quite large printing credit of faculty to also hold for the printers of the central administration of the university for example.

Example 5.4.6 (Control)**.**
$R^1_{Prop} = PrinterInLaboratoryRealm(Entity)$
$C^{Policy}_x = R^1_{Prop}$

$C^{Policy}_t = R^2_{Prop}$
$R^2_{Prop} : Faculty(Subject) \rightarrow PagesLEQ2000(Subject)$

Violation



The violation has a rule, which when evaluated to true enforces a policy recovery. Several *Violations* can be modeled for a Policy, each of which maps an evaluation of the Rules in the Policy Control part to a given violation value.

Violation Value and Violation Type

The violation values are defined in a violation type. A violation type is a set of violation values. For example, the following enumeration: {Red, Orange, Yellow, Blue, Green, Gray} defines a violation type called *traffic lights*[7]. Each of the enumeration values constitutes a violation value (e.g., Red).

Figure 5.7: Violation Type - Lights - Discrete Set of Violation Values

Violation Decision

The CoReL models do not provide functionality to model complete decision making. The lacking modeling functionality is the mapping between the different possible valuations of a policy's controls and the corresponding violations. Violations are defined directly for a policy. This decoupling between violations and controls allows to reuse CoReL models for defining different decision-making tactics by modifying violation decisions for the same policy. We define *Violation Decision* here:

Definition 5.4.5 (Violation Decision)**.** *A* Violation decision *is a mapping function between the set of controls and the set of violations.*
$ViolationDecision : 2^{Controls} \mapsto 2^{Violations}$.
It is made out of two functions ViolationValuation *and* ViolationMapping*:*
$ViolationValuation : 2^{Controls} \mapsto ViolationValues.$
$ViolationMapping : 2^{ViolationValues} \mapsto Violations.$

[7]This is actually a semantic violation as normally traffic lights define 3 discrete values, not 6.

A *Violation Diagram* defines two mappings: (i) from combinations of control valuations (i.e., truth valuations of the control rule) to a violation value, and (ii) from a (set of) violation values to a violation. The violation diagram is application specific and is built by CoReL policy modelers. In the CoReL tool (cf. Chapter 7), we implemented one example of a discrete violation type, called the *traffic light type* in Figure 5.7. More complex types of violation types and corresponding valuation functions can be defined by relying on utility theory from economics (as used previously in [LAS+06, SL05, LS06]), and this has already been used to define expressive business policies which use multiple valuation modeling strategies for policies such as continuous functions and fuzzy logic [EK07].

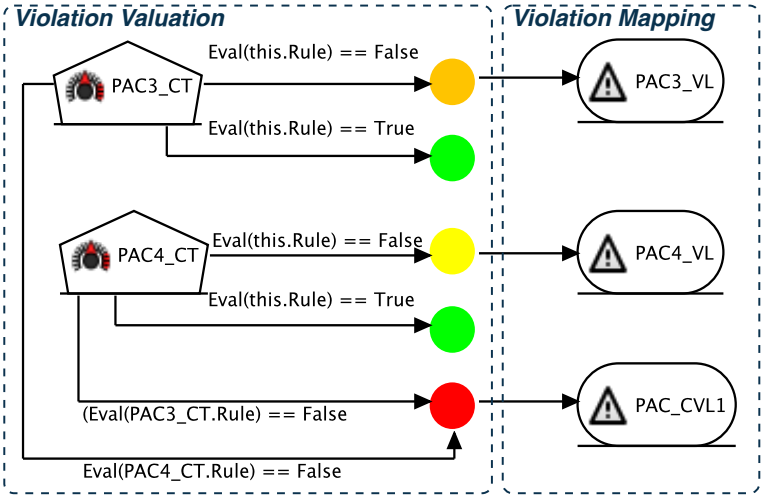


Figure 5.8: Violation Diagram

Le us consider a simple example, shown in Figure 5.8. Take a policy P, let the policy have two controls. The violation type used with P is defined by the traffic light type. The user may model a Violation of value *Orange* if the control $PAC3\_CT$ rule returns False, and $Yellow$ if the control $PAC4\_CT$ rule returns False. If either one of the control rules is evaluated to true, we compute the *Green* violation value.

For this example, we decide not to map the *Green* valuations to any violation (e.g., for modeling rewards). Note that according to our function-based definition of violation mappings, we constrain the violation model to map any of the two *Green* violation values to the same violation, as these two shapes in the diagram are in fact two occurences of the same violation value. We may map the *Orange* violation value to the $PAC3\_VL$ violation, and the $Yellow$ violation value to the $PAC4\_VL$.

In theory, we would be better off modeling a mapping from all possible control rule truth valuations to a violation value, but we decided not to force a complete modeling of this mapping in violation diagrams, in order to quicken the modeling process.

An example of expressing complex valuations is the valuation to a *Red* violation value in case each rule in both controls are evaluated to $False$. Finally, we may map the *Red* violation value to a complex violation called PAC_CVL1.

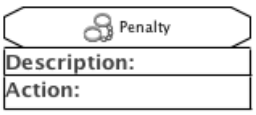
The reader will have noticed two important aspects of our design choices. First, the choice of the violation type is crucial. In terms of expressiveness, the smaller this violation type is, the more limited the number of violation mappings we may express. Second, we could easily make the violation decision function more expressive by proceeding to two modifications of CoReL: (i) defining the domain of this function to be the control rules instead of controls, and by (ii) allowing a control to have several rules. We voluntarily made the choice of limiting a control to a single rule for the sake of avoiding non-essential complexity in introducing the results of this work, as such an extension of CoReL is trivial.

5.4.1.4 Recovery Block

After computing a violation value, a function will decide whether to and how to recover from the Violation. The Policy must also produce an execution decision (e.g., *Resume/Stop*) to send out to the business process execution engine.

Before a decision is taken with regard to the process execution, a corrective action can be taken after the violation has been identified. For each Violation, CoReL allows to specify a step called Violation Recovery which triggers additional corrective actions. To this purpose, we distinguish different types of Violation Recoveries: Handling, Compensation and Reparation.

Handling



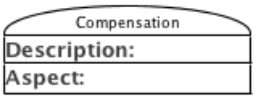
Every Violation has a (possibly empty) set of *Handlings* used to react to the violation. A Handling always triggers some specific informative action, such as the execution of a function that calculates penalty points, or an action that sends notification emails. A violation represents a deviation from ideal behavior, as expressed by the Control in the policy. The deviation might be regarded as positive and therefore is encouraged, or as negative and therefore must be discouraged. 'Positive' Violations are named *Reward* and 'negative' violations are named *Penalty*. In the remainder of the thesis, we may use the word *Sanction* to refer to *Penalty*, the two terms are equivalent. Both Sanctions and Rewards are special types of Handlings (indicated by inheritance in the metamodel of CoReL in Figure 5.5).

To the policy modeler, Rewards are used to model how to react positively to the Violation, and Sanctions, on the opposite, are used to model negative reactions. Note that in CoReL, Rewards and Sanctions are modeled in the same way, the only concrete syntax difference is the color of the symbol lines: red for Sanctions and blue for Rewards. One could see the sanction and the reward as two different modalities of violation expressions. This means a violation can at the same time define a reward and a sanction. This modeling freedom increases the expressiveness of the framework.

For example, a policy modeler can assign a Reward to a Violation with value *Green* that increases some rating given to the user of the printer. Alternatively, the policy modeler can assign a Sanction to the printer user that decreases his rating if the Policy decides that he caused a *Red* Violation. The actions used in the definition of Handling are application-specific, provided by the business model and are to be selected by the CoReL policy modeler.

A Handling can directly trigger the checking of another policy, thereby bypassing the context-based activation of a policy. This is the way CoReL models decision propagation.

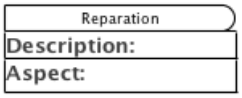
Compensation



A Compensation is a corrective action that introduces no change to the system (i.e., the enterprise or process model). Unlike a Handling, it does not operate on system entities such as users, user accounts etc. This is the subtle conceptual difference between handlings and compensations. Concisely put, a Compensation is a corrective action operating on the outside boundary of the system. It is neither a Reward nor a Sanction.

A classical example is that of a pharmaceutical company whose drugs have harmed patients. A Compensation might be a financial subsidy and/or taking in charge the medical treatment of the damages caused by the drug.

Reparation



Apart from informative reactions which are modeled by a Handling, a *Policy* can trigger the execution of an action that will modify the system. This action is undertaken by a third party (i.e., not the policy engine) and in effect, modifies the enterprise model, for example modifying a business processes.

Reparations specify a controlled modification of the business process model as already proposed in similar fashion in Namiri's work [Nam08]. Reparations are meant to *allow the user to model a modification to the enterprise model* that he thinks will make a *violation less likely to occur again*. Intuitively, the use of reparations would make more sense at run-time as a mechanism to define adaptive business processes. Here, one could distinguish (one-time) changes made to a running instance of a business process from (permanent) changes made to the process model itself, thereby impacting all future instantiations of the latter. In this work, we consider the second scenario.

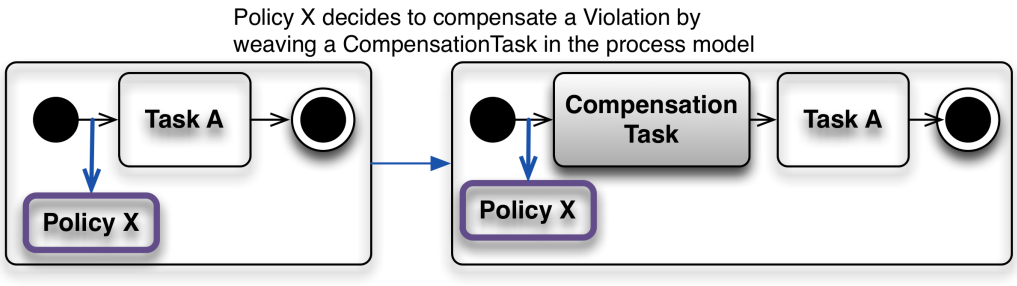


Figure 5.9: Process Compensation Illustration

A Reparation action might be implemented as a process task that can be weaved into the business process model. Figure 5.9 illustrates the concept of Reparation in process models on a state chart. The difference to a Violation Handling is that the actions that may be triggered by the latter are not corrective actions as is the case for Reparations.

The semantics of Reparations are business modeling language specific and undergo some consistency rules. One of such rules is that the join point where the Reparation action is weaved into a business process cannot be reachable from the current active task in the execution of a business process .

5.4.1.5 Wrap-Up

In this section, we gave a first introduction to the CoReL policy modeling language. We listed the most important concepts from the abstract syntax model, which are necessary to understand the functionalities supported by the language. We also introduced the semantics of the language informally, illustrated by a small example.

5.5 Informal Semantics of CoReL Policy Interpretation

5.5.1 Policy Governance

We say that a policy governs the behavior of a system, i.e., of a business process. It does this by governing an ASE triple instances in the process (see Figure 5.10).

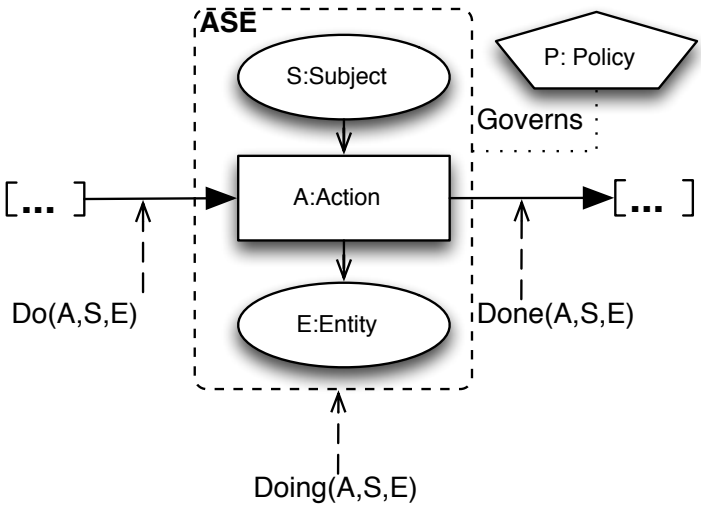


Figure 5.10: ASE Triple - Predicates

A CoReL policy uses mechanisms to govern a business process execution. These mechanisms have been introduced in the previous chapter. The most important ones are the Context, Control and Violation concepts. We explain how these constructs are used by CoReL. The formal semantics can be found in Sections 6.1.1 and 6.1.1.

5.5.1.1 Policy Activation

A context defines when a Policy becomes Active. This is illustrated in Figure 5.11, where S stands for System (e.g., the business process), and P stands for Policy.

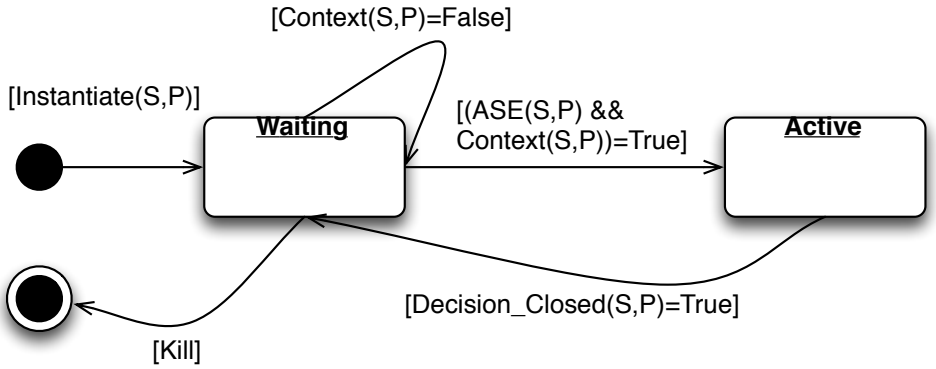


Figure 5.11: Policy State Machine - Context Semantics

After being instantiated, a Policy becomes ready for activation. When active, a policy will make decisions. A Policy will become Active only if the ASE on which it applies is being executed. Another condition is that the Context of thePolicy must be evaluated to True on the System and the Policy. The P (Policy) parameter here allows to access all information about the policy such as the ASE.

When the Policy has taken its decision and the decision has been implemented, we say the policy decision is closed. In this case, the Policy returns back to its passive Waiting state.

5.5.1.2 Policy Decision

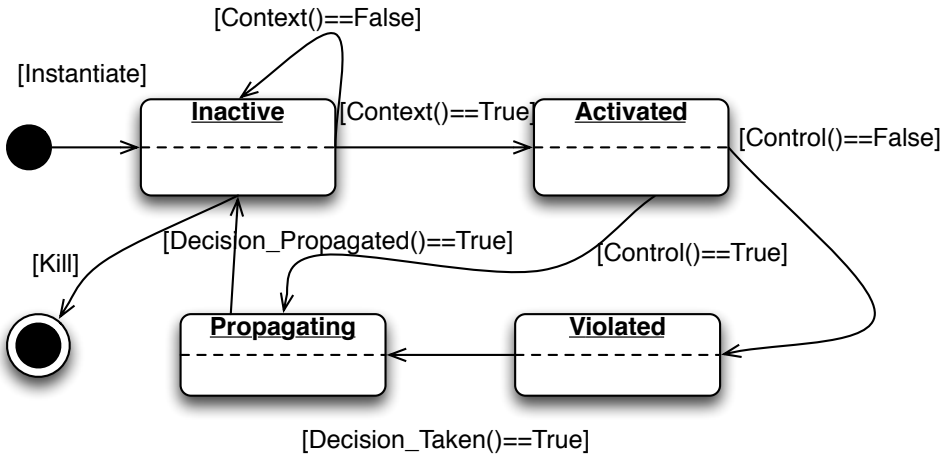


Figure 5.12: Policy State Machine - Decision Making & Propagation

In Figure 5.12 we explain what is meant by *Policy Decision*. Once the Policy is active, it will decide whether there is a violation or not, depending on the boolean valuation of the Control. In fact, this definition is only adequate if we have a single Control for a Policy. In the most general case, we have a Violation function which computes a violation value out of the valuation of all the Controls.

This step is followed by a decision-making step where all the required actions that must be taken are computed, as well as whether to resume the execution of the process or interrupt it.

So the decision is the tuple made of the execution decision, and the recovery actions to be taken. Once a decision is taken, it must be enforced. We call this *propagating* the decision. Once the decision has been propagated, the Policy goes back into an inactive state.

5.5.2 Enforcement of Authorization Policies

After explaining how the policy activation and decision mechanisms work with policy modeling constructs, this section shows how the enforcement of authorization policies is done.

5.5.2.1 Single Violation Authorization

Authorization policies are either Permission or Prohibition. Authorization policies are so-called punctual policies, which means that the policy compliance decision is made at the same time the policy is evaluated. That is why authorization policies are simpler in nature compared to other types of deontic modalities. The behavior of a permission policy is explained in Figure 5.13. Note that the behavior of a Permission policy is symmetric with the behavior of a Prohibition policy since one can be expressed in terms of another: $Permission(Doing(a, s, e)) \triangleq (notProhibition(Doing(a, s, e)))$.

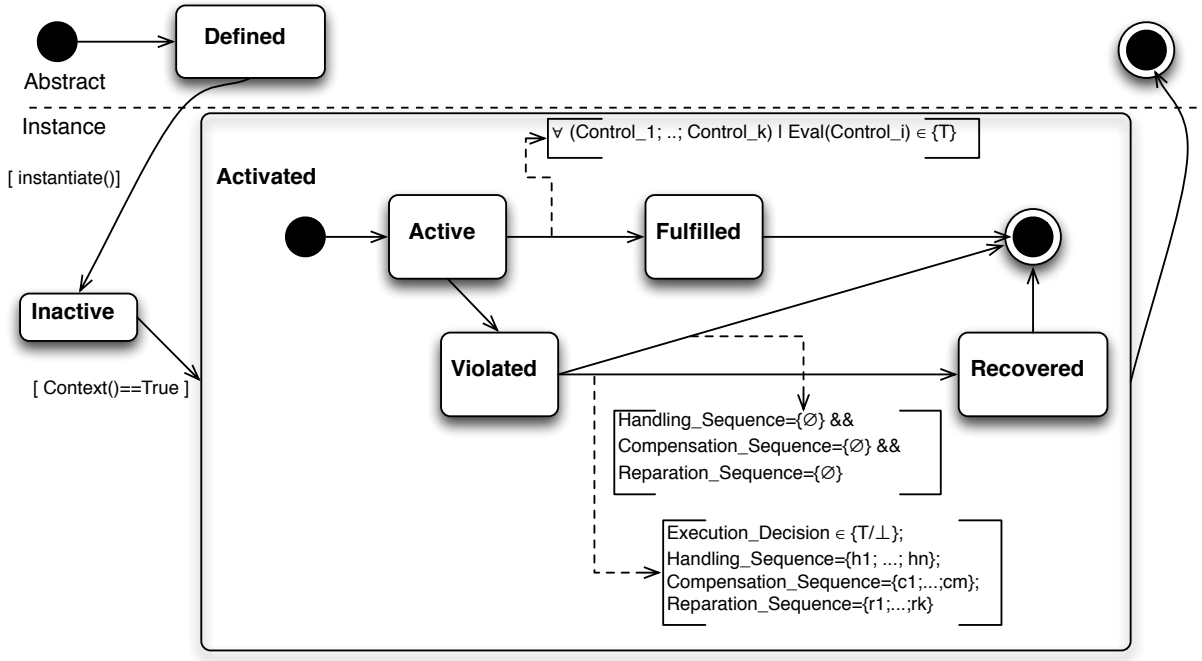


Figure 5.13: CoReL Policy - Permission State Machine - Single Violation

After being activated, the policy is either violated or fulfilled. In case it is violated, it must be recovered immediately if a recovery is defined. Otherwise, the policy stays in the violated state and terminates its own execution carrying an appropriate execution decision.

5.5.2.2 Multiple Violation Permissions

The previously introduced state machine is only valid in the case where we assume we consider a single violation. However, one of the peculiarities of CoReL is to support multiple violations.

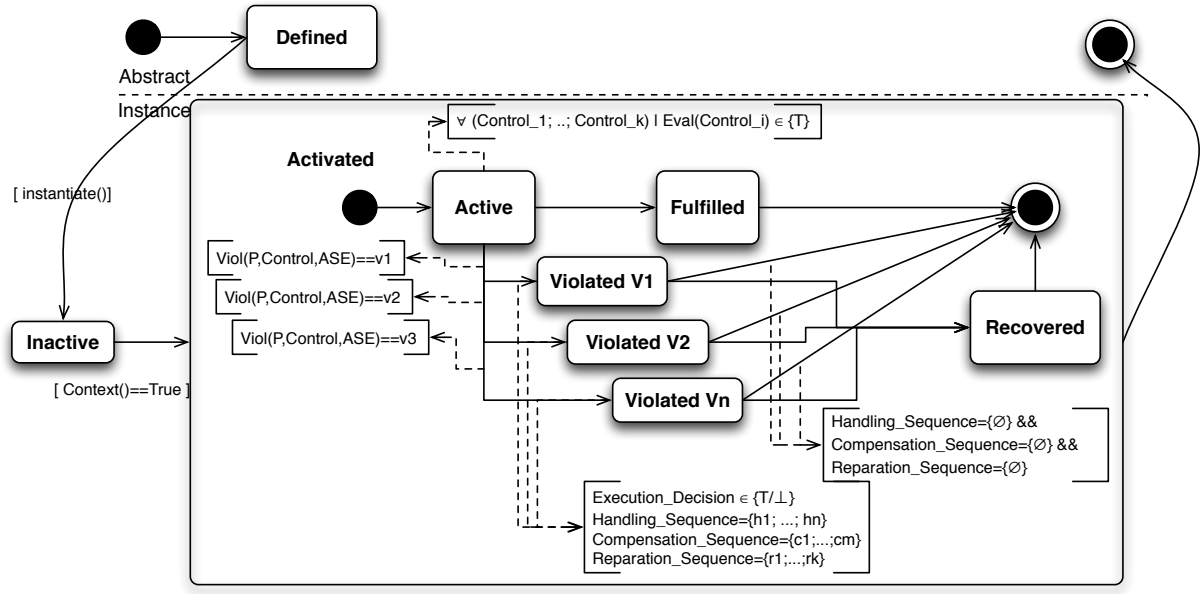


Figure 5.14: CoReL Policy - Permission State Machine - Multiple Violations

Figure 5.14 shows the state machine for this case. The only difference to Figure 5.13 is that there are multiple state transitions from the active state to a given violation state. These transitions are decided by the violation function.

5.6 Modeling A Simple Business Process Printer Management Regulation

Table 5.3 gives the CRs defined by the system administrator for controlling the use of the university printers. The various CoReL elements can be reused (e.g., an *ASE-Triple* or a Violation reused by several Policies). We will begin by creating a regulation diagram, and follow by creating CoReL policy diagrams implementing each of the policies in the regulation diagram.

Table 5.3: Compliance Requirement Examples

CR	Definition
CR1	External students cannot print on university printers. Regular and exchange students can.
CR2	Students are prohibited from printing on the week-end. Faculty members can.
CR3	Students are limited to 400 pages a month, with a bonus of 50 pages if it is their first month at university. Every student who prints less that 30% of his allocated print pages over a whole semester can receive a 5 euro waiver ticket to use the university's swimming pool.
CR4	faculty members can print up to 2000 pages a month. Faculty members are allowed a violation of the max number of pages if it amounts to less than 10%, in which case they are warned per email. In case they exceed the number of allocated pages by more than 10% pages, their access to printers is stopped.

Out of this set of compliance requirements extracted from a textual source, we create the regulation model shown in Figure 5.16. The regulation model simply lists all compliance requirements linking each one to the policy implementing the CR.

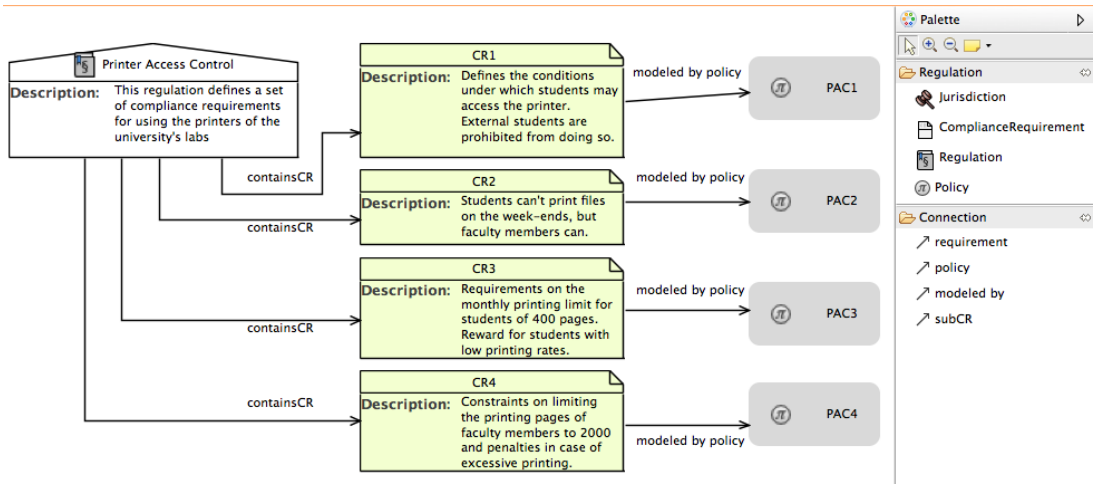


Figure 5.15: CoReL - Printer Example - Regulation Model

In Figure 5.17, we model the first CR. These diagrams are accessible by right-clicking on the policy in the regulation model and selecting *'Jump To Policy'*. This opens the CoReL policy model assigned to the policy object in the regulation model. Only one policy model can be assigned at a time, but the tool allows to change the policy model assignments.

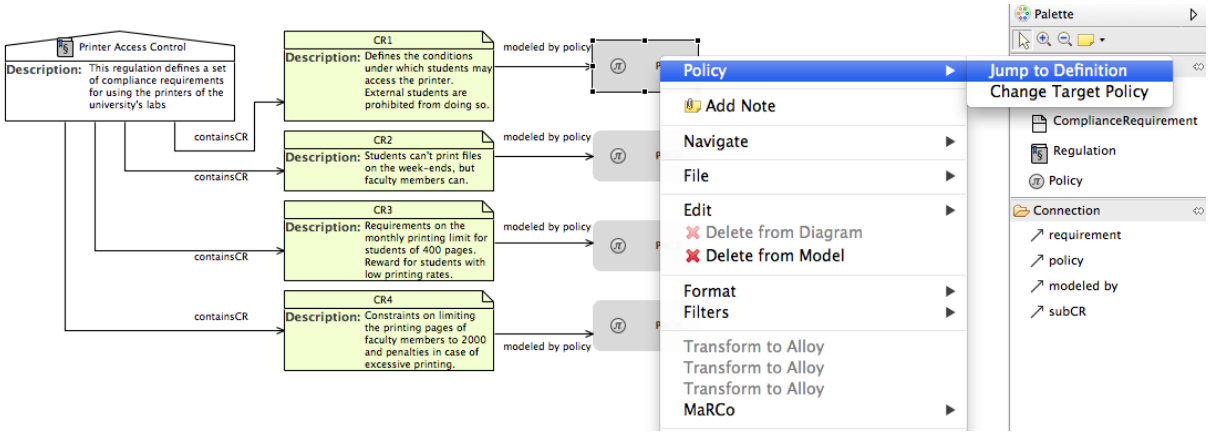


Figure 5.16: CoReL - Printer Example - Accessing an Assigned Policy Model

In Figure 5.17, we model the first CR. The diagram describes that in the context where a printing action is executed by a user who is an internal student, the control is always true and therefore a permission to execute the action is decided by the policy, The small green symbol on the policy shape describes the deontic modality of the policy and in this case tells the user that the decision of the policy is going to make is a permission.

However, another more concise and elegant way of modeling this same policy is given in Figure 5.18. In this model, the context is absent, which means it does not need to be checked.

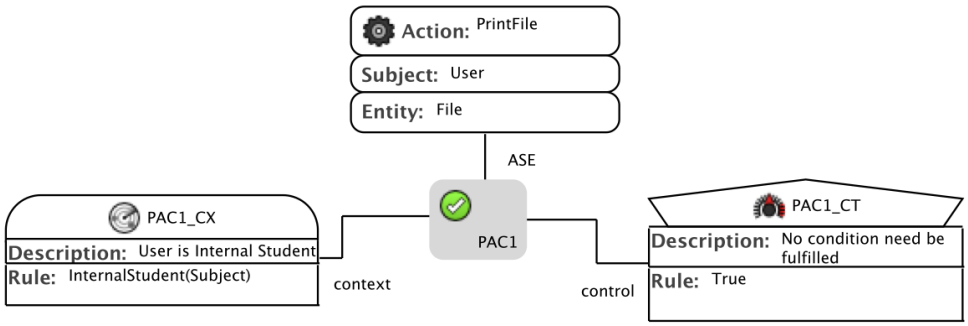


Figure 5.17: CoReL - Printer Example - Modeling CR1

The context is therefore evaluated to be holding all the time. We change the control to express the constraint that the student must be internal. We keep the same permission deontic modality for the policy. The ASE will stay the same across all the policy diagrams in this printer example.

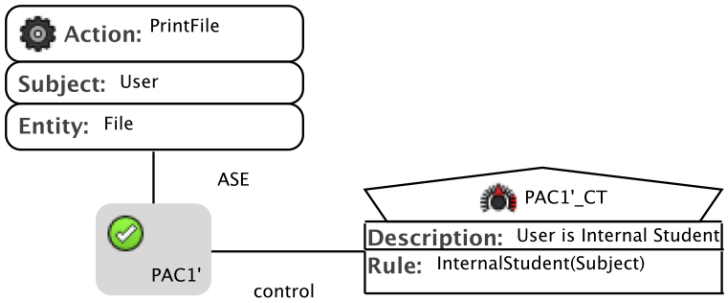


Figure 5.18: CoReL - Printer Example - Modeling CR1 - second alternative

We see that the previous policy is just one decision element described in the CR1. We need to model the other part, which concerns external students, in Figure 5.19. This policy simply says that in all contexts, when the control part, stating that the subject is currently an external student, holds, then we must enforce a prohibition to execute the ASE.

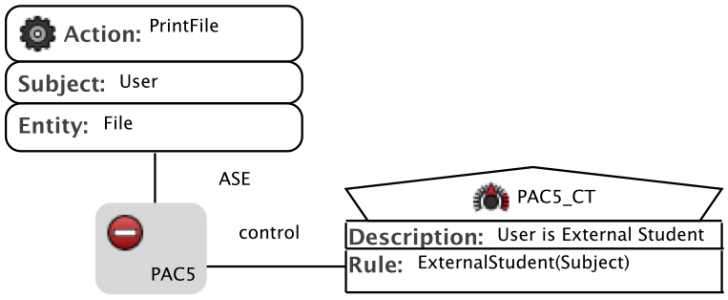


Figure 5.19: CoReL - Printer Example - Modeling CR1 - External Students

In order to model CR2, we need to use the same ASE as previous, define a context holding only on the week-ends, and define as a control the constraint that the subject is not a faculty member. By setting the deontic modality of the policy PAC2 to be a pro-

hibition, we guarantee that when the control holds the policy will decide not to resume the process.

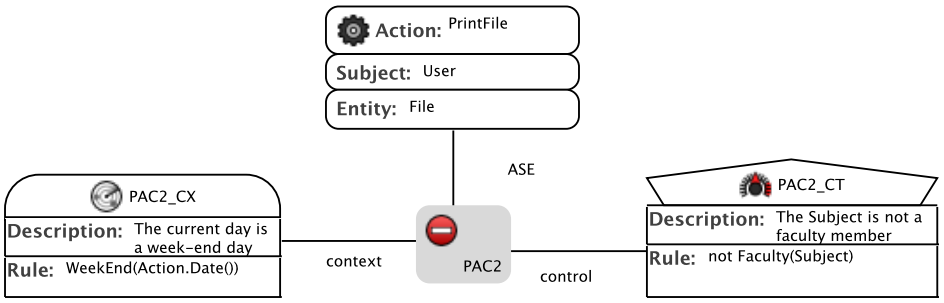


Figure 5.20: CoReL - Printer Example - CR2

In fact, this policy is not an accurate implementation of the CR, simply because it uncovers an implicit assumption made when interpreting the CR, which is that we only have student and faculty users, and therefore, only allowing faculty means not allowing everyone else. It is important to notice that we do not define a violation here, which means that the decision of the policy must be enforced immediately.

This means the process will be aborted if the user currently printing on the week-end is a faculty member. This is obviously not what we want. What we would like to model is the following statement: *'when the user currently printing is not a faculty member, then this cannot be on a week-end day'*. If we set the context to hold when the user is not a faculty member, and the control to hold when the printing happens on a week-end day, we can express the desired decision-making we want. We model this in Figure 5.21.

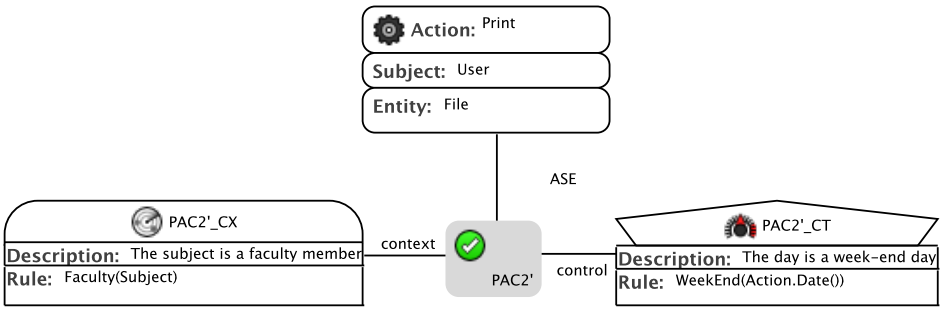


Figure 5.21: CoReL - Printer Example - Corrected CR2

The CR3 is more complex than CR1 and CR2 since it specifies violations as well as violation handlings, see Figure 5.22. In case the control (constraining the number of extra printing pages) is violated, we must enforce one of the two specified violations each having a handling as a recovery. Both recoveries in this case are rewards for low printing. One missing violation is when the user prints more than 400 pages. Since there is no violation defined for this case, it is not allowed, and the policy will not take the decision to permit the ASE to happen, it will abort the process.

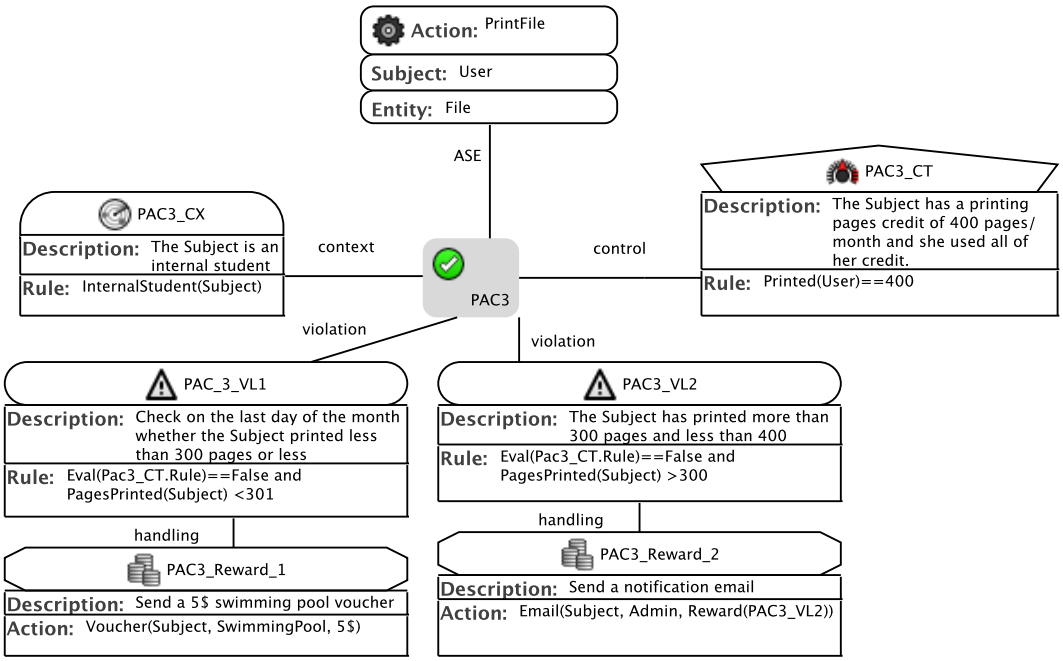


Figure 5.22: CoReL - Printer Example - CR3

Finally, CR4 is different in that it defined a single possible violation with two distinct sanctions (see Figure 5.23). Both sanctions are enforced concurrently and can be implemented outside the policy engine as system calls.

In this section, we illustrated some of the expressiveness of CoReL using a set of simple compliance requirements for printer management. However, many questions are still raised as to the formal semantics which describe policy enforcement using CoReL. The following chapter 6 will answer open questions regarding the formal semantics for policy interpretation and enforcement.

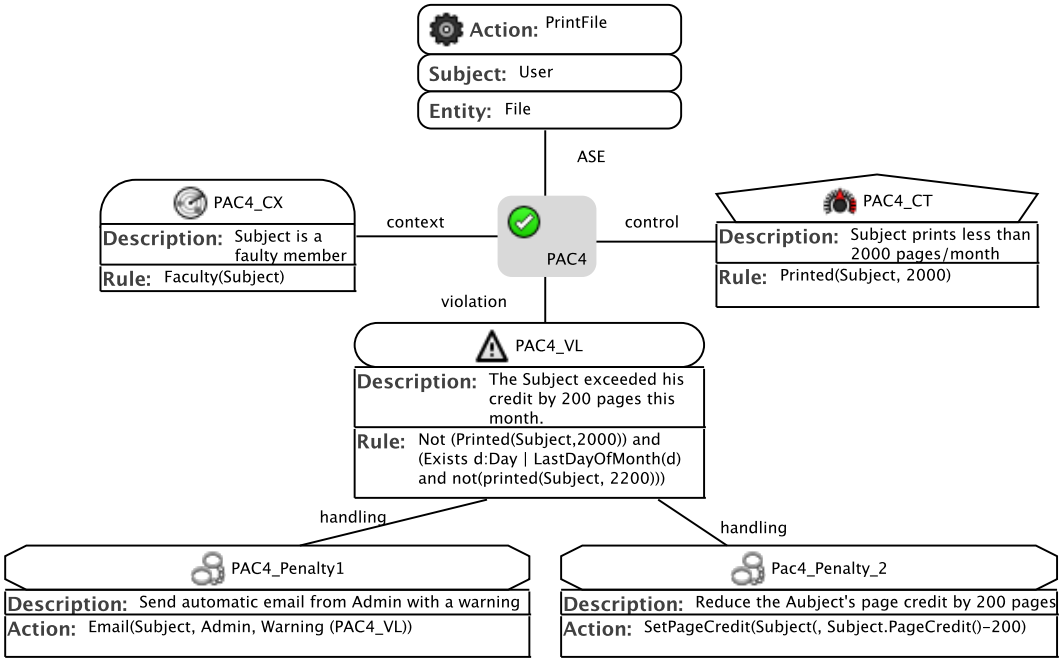


Figure 5.23: CoReL - Printer Example - CR4

Chapter 6

Semantic Foundations of CoReL

My very conscious desire to reduce the demands made on reasoning is undoubtedly the result of my professional experience as a programmer, but the seed has been sown nearly thirty years ago, when I received from my mother the shortest and wisest lesson in Mathematics. Being asked by me what "Trigonometry" was all about and whether it was difficult, she answered: "Oh no, it is not difficult: know the formulae, and whenever you seem to need more than five lines for a problem, try something else, for in all probability you are on the wrong track.

E. W. Dijkstra. Homo Cogitans - A small study of the art of thinking. 1975.

In this chapter we introduce a formalization of the metamodel of the CoReL language by giving the language operational semantics. This formalization is done declaratively using the Alloy language. For background on the Alloy formalism used please consult Section 2.3.

6.1 Formalizing CoReL

6.1.1 Formalizing the CoReL Abstract Syntax Metamodel

The Rule Module

This very simple module is used to model CoReL policy rules. For a rule, we need to know the language it is written in, the formula that represents the rule constraint, as well as the target objects. However, for pure policy evaluation in a declarative formalism, these constructs are not compulsory.

```
445 module rule
446
447 open ase
448 open util/boolean
449
450 sig Rule{
451   eval: Bool
452 }
453 pred Rule::holds{
454   this·eval·isTrue
455 }
```

We can model Rule evaluation using a boolean field called eval. Unless we specify it to behave otherwise, the Alloy engine will generate a state space where the value of this field can be either True or False. This can be used in the general state space exploration case. Then, we model a

predicate holds for Rules which just returns True in case the boolean value of eval is equal to True.

Contrarily to the CoReL metamodel in Chapter 5, in this Alloy module, there is no need to specify the rule statement as this is application-specific. In concrete policy examples in Alloy, we will show how to define such application-specific rules. There is also no use in specifying the formalism used for the rule, as we will use Alloy expressions in first order logic to implement rules in our formalization. We do not claim that rules written in first order logic are enough to express the wide range of compliance requirements we consider in this thesis; we will show that combining rules in first order logic with the semantics defined for CoReL allow to express a combination of deontic modalities and first order statements.

Policy Constructs

The CoReL module given below requires to import the ase, sbp and rule Alloy modules. Then the abstract signature for Policy is declared with the following relations: governs which relates to an ASE, context and control, which both relate to a Constraint (we will explain this signature in the next paragraph), and a violation field. We follow in this signature the structure given by the abstract syntax metamodel of CoReL.

```
456 module corel
457
458 open ase
459 open sbp
460 open rule as fl
461
462 abstract sig Policy{
463   governs: ASE,
464   context: Constraint,
465   control: Constraint,
466   violation: Violation
467 }
```

Policy Governance

Here, we will declare some useful Policy predicates which are needed for the state machine semantics of Policies. The governs(ase) predicate says that a given ASE is governed by the policy.

```
468 pred Policy::governs[a:ASE] {
469  a in this·governs
470 }
471 pred Policy::applies[s:SBP] {
472  some a:ASE | a in sbp/getAses[s] and this·active[a]
473 }
```

The last predicate declaration is called applies(SBP) and tells whether a Policy currently is applying (i.e., must be evaluated) in a given SBP process. A Policy applies means its Context and Control parts (the Constraints) must be evaluated. A Policy is said to apply in a business process whenever it governs an ASE, and this ASE is currently being executed (i.e., is in an active state).

Policy Evaluation

The Policy evaluation is conducted when evaluating the constraints expressed by policy Rules. This is needed when evaluating Controls and Contexts. For this we use a signature Constraint

which embodies a sequence of Rules. We define the holds() predicate that tells when a constraint is complied with. It simply means that all Rules in the sequence hold (see the holds() predicate of Rule).

```
474 sig policy_eval{}
475
476 abstract sig Constraint {
477   rules: seq Rule
478 }
479 pred Constraint::holds{
480   all r: this·rules·elems| r·holds
481 }
```

Policy Violation

We model the Violation for the simple case of a single violation. We denote *dv* for default violation. We have two predicates defined on a Policy: violates_dv(ase) resp. violates_dv(sbp) which describes the state when a Violation *dv* is computed for a given ASE resp. a given *SBP*.

The implementation of violations is application-specific and will be illustrated in Chapter 9 in a case study. However, the definition of a default violation in CoReL is straightforward. When the ASE is active, and the context holds but the control does not, we have a violation. The second predicate checks in the SBP for any ASE which is violated and concludes that the SBP is in a state of default violation too.

```
482 pred Policy::violates_dv[a:ASE]{
483    this·active[a] and this·context·holds and not this·control·holds
484 }
485 pred Policy::violates_dv[sbp:SBP]{
486   some a:ASE | this·violates_dv[a]
487 }
```

Policy Decision

We define two basic execution decisions called resume and abort, the names are self-explanatory. We also model the decision() function which returns the decision on execution by the Policy. In the following Alloy code snippet, we illustrate the *decision* function by defining the positive default decision that the process should resume execution normally no matter whether we have a violation or not.

```
488 abstract sig PolicyDecision{}
489 one sig resume, abort extends PolicyDecision{}
490
491 fun Policy::decision: PolicyDecision{
492   resume
493 }
494
495 abstract sig ExecutionDecision {}
496 one sig Abort, Resume extends ExecutionDecision {}
497 }
```

6.1.2 Formalizing the CoReL Execution Semantics

In this section, we define the semantics of Policy enforcement. We define how the states of a policy are calculated during the execution of a sbp.

CoReL Policy States

The first step in the formalization is to define an abstract signature PolicyState. We also define the following possible concrete policy states: inactive, active, fulfilled, violated, weaklyFulfilled, recovered.

```
499 module CoReLexec
500 open corel
501 open SBPexec
502
503 abstract sig PolicyState{}
504 one sig inactive, active, fulfilled, violated, weaklyFulfilled, recovered extends PolicyState{}
```

Policy States

The signature PolState defines a mapping from all Policies to their current PolicyState defined above. We also define two Alloy functions on PolState: *state(Policy)* returns the current state of a given Policy, and *policiesInState(PolicyState)* returns all Policies which have as a current state the PolicyState (e.g., active) given in parameter.

```
505 some sig PolState{
506   pstate: Policy →one PolicyState
507 }
508 fun PolState::state[p: Policy]: PolicyState{
509   p·(this·pstate)
510 }
511 fun PolState::policiesInState[s: PolicyState]: set Policy{
512   (this·pstate)·s
513 }
```

Compliance State

Here we define the overall state of the system, in terms of compliance, which we conveniently call *ComplianceState*. The *ComplianceState* is a composite state of both the *ProcessState* (the state of the SBP as defined in *sbp_ exec* in Section 4.4) and the *PolState* (the state of all *Policies*).

The system's (business process's) execution semantics are defined in terms of a history of *ComplianceStates*. We have to initialize the system's execution with an initial *ComplianceState* called *cInitState*. In this state, all *Policies* are *inactive* and the process is in the initial state aInitState (where the only active node is the *Start* node).

```
514 some sig ComplianceState{
515   processState: ProcessState,
516   polState: PolState // states of policies
517 }
518 one sig cInitState extends ComplianceState{}{
519   processState = aInitState
520   all p: Policy| policyState[p] = inactive
521 }
522 fun ComplianceState::policyState[p: Policy]: PolicyState{
523   (this·polState)·state[p]
524 }
525 fun ComplianceState::activePolicies: set Policy{
526   (this·polState)·policiesInState[active]
527 }
```

Two functions are defined on the signature *ComplianceState*. The function *policyState(Policy)* returns the state of a given *Policy* given in parameter. The function *activePolicies()* returns the set of all *Policies* which are active.

Consistency of Compliance States: Process States
We have to guarantee that we have no orphan process states. This means, there exists no *ProcessState* which is not in the *processState* field of the *ComplianceState*.

```
528 fact noOrphanProcessState{
529   ProcessState in ComplianceState·processState
530 }
```

State Transitions - Compliance State Transition Machine
For every state transition between the state signatures defined above, we model two predicates for the Policy signature.

The first predicate's name follows the pattern condBecomes<NameOfState>(ComplianceState). This predicate is used to model the transition guard. For instance, for the transition guard *condBecomesActive*, we say the guard becomes True *if and only if* the Policy is currently inactive and there exists an ASE which is governed by the Policy in the SBP which is currently active.

The second predicate's name follows the pattern *becomes<NameOfState>(ComplianceState, ComplianceState)*. This predicate models when a Policy actually switches states from a *ComplianceState* to another *ComplianceState'*. For example *becomesActive(ComplianceState, ComplianceState)* models the state transition from inactive to active.

The same explanations hold for the guards and transitions:

- From active to fulfilled. The transition is enabled (guard is $True$) when the Policy is active and the Context as well as the Control hold.
- From active to violated. The transition is enabled when the Policy is active and the Context holds but the Control does not hold.
- From active to weaklyFulfilled. The Transition is enabled when the Policy is active and the Context does not hold.

```
531 pred Policy::condBecomesActive[s: ComplianceState]{
532     s·policyState[this] = inactive and some a:ASE | (a in s·processState·activeNodes·se and this·governs[a])
533 }
534 pred Policy::becomesActive[s,s':ComplianceState]{
535   s·policyState[this] = inactive and s'·policyState[this] = active
536 }
537
538 pred Policy::condBecomesFulfilled[s: ComplianceState]{
539     s·policyState[this] = active and this·context·holds and this·control·holds
540 }
541 pred Policy::becomesFulfilled[s,s':ComplianceState]{
542   s·policyState[this] = active and s'·policyState[this] = fulfilled
543 }
544
```

```
545 pred Policy::condBecomesViolated[s: ComplianceState]{
546     s·policyState[this] = active and this·context·holds and not this·control·holds
547 }
548 pred Policy::becomesViolated[s,s':ComplianceState]{
549   s·policyState[this] = active and s'·policyState[this] = violated
550 }
551
552 pred Policy::condBecomesWeaklyFulfilled[s: ComplianceState]{
553     s·policyState[this] = active and not this·context·holds
554 }
555 pred Policy::becomesWeaklyFulfilled[s,s':ComplianceState]{
556   s·policyState[this] = active and s'·policyState[this] = weaklyFulfilled
557 }
```

Policy State Transitions

The predicate *condTransition(ComplianceState)* defined on *Policies* tells whether a given state transition is enabled. While the predicate *stateUnchanged(ComplianceState, ComplianceState)* also defined over *Policies* tells whether the state of a *Policy* remained unchanged between two different *ComplianceStates* s and s'.

```
558 // some transition is possible for this policy in state s
559 pred Policy::condTransition[s: ComplianceState]{
560   this·condBecomesActive[s]
561   or this·condBecomesFulfilled[s]
562   or this·condBecomesViolated[s]
563   or this·condBecomesWeaklyFulfilled[s]
564 }
565
566 pred Policy::stateUnchanged[s,s':ComplianceState]{
567   s·policyState[this] =s'·policyState[this]
568 }
```

Compliance State Transitions

The predicate *complianceStep(ComplianceStep, ComplianceStep)* models the transition from one *ComplianceState* to another. *ComplianceState* transitions work along the following principle: every time a *Policy* can enact a policy state transition as part of its decision making process, it has precedence and no process state transition is executed until all active policies have reached a locking state where no policy state transition is possible anymore. Once this state is reached, process state transitions are executed as defined in the *sbp* model, until a new *Policy* state transition becomes possible again, and thus the process execution is halted once more.

The predicate works as follows:

- If there is no *Policy* state transition that is applicable then we have a state transition in the process states as defined in the Alloy module *sbpexec* while all policy states remain unchanged.
- Or else, there is at least one policy state transition enabled. In this case, we ensure that all policy state transitions which are enabled are actually executed.
- In order for the model to work properly, we must add the fact that no policy state transition is executed if and only if no policy state transition is enabled. Without this latter statement,

we might obtain model instances where we have policy state changes although no policy state transition was enabled.

```
569 pred complianceStep[s,s': ComplianceState]{
570   (no p:Policy | p·condTransition[s])// no policy transition applicable
571     implies (stateStep[SBP, s·processState, s'·processState]
572       and all p:Policy| p·stateUnchanged[s,s']
573     )
574     else // at least one policy transition applicable
575       (s·processState = s'·processState and
576         (all p:Policy |
577         ((p·condBecomesActive[s] iff p·becomesActive[s,s']) and
578         (p·condBecomesFulfilled[s] iff p·becomesFulFilled[s,s']) and
579         (p·condBecomesViolated[s] iff p·becomesViolated[s,s']) and
580         (p·condBecomesWeaklyFulfilled[s] iff p·becomesWeaklyFulfilled[s,s']) and
581         (not p·condTransition[s] iff p·stateUnchanged[s,s'])
582       )))
583 }
```

Compliance Trace - System Execution Semantics

As said earlier, we define the execution semantics of a business process as a trace of *ComplianceStates* embedding the process and policy state transitions. This is illustrated in Figure 6.1, where we have look at one example of a *ComplianceTrace* made of *ComplianceStates*. Each *ComplianceState* is composed of the *ProcessState* and the *PolState*.

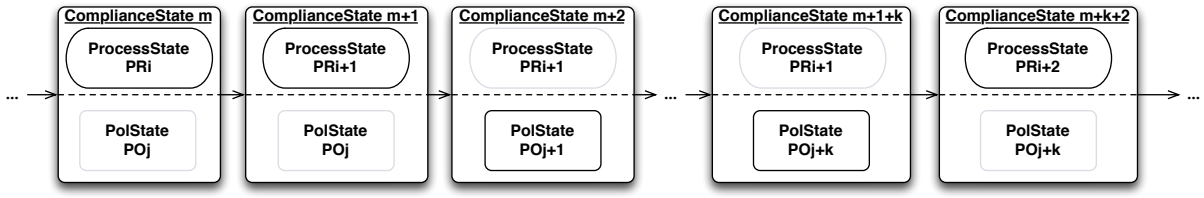


Figure 6.1: Compliance Trace - Illustration

We define the singleton signature of *ComplianceTrace* as a sequence of *ComplianceStates*. We add as signature facts that the first *ComplianceState* must be the special state named *cInitState*. We also add that each two adjacent *ComplianceStates* must be linked to one another by *compliance steps*, as defined in the *ComplianceStep* predicate above.

```
584 one sig ComplianceTrace {
585   states: seq ComplianceState,
586 }{
587   states·first = cInitState
588   all i: (states·inds – states·lastIdx)| complianceStep[states[i], states[add[i,1]]]
589 }
590 pred ComplianceTrace::valid[i: Int]{// compliance trace valid up to state i
591   all j: (this·states)·inds | lt[j,i] implies complianceStep[this·states[j], this·states[add[j,1]]]
592 }
```

Finally, we add the predicate which tells whether the *ComplianceTrace* is valid up to the index equal to the *Int* value given as parameter. This predicate simply verifies that for every two consecutive states in the *ComplianceTrace* sequence, whose indices are smaller than the *Int*

parameter, the *ComplianceStep* predicate holds. That is, as illustrated in Figure 6.1, these two consecutive *ComplianceStates* are linked with an arrow.

Helper Functions - Compliance Trace

Again, we define here some functions useful for writing other parts of the formalization. The *processStates()* function returns all the *ProcessStates* contains in the *ComplianceTrace*. The *lastState()* function returns the currently last *ComplianceState* in the *ComplianceTrace*.

```
593 fun ComplianceTrace::processStates: set ProcessState {
594   this·states·elems·processState
595 }
596 fun ComplianceTrace::lastState: ComplianceState {
597   this·states[this·states·lastIdx]
598 }
```

Consistency of Compliance Traces

We also define a *noOrphanStates* fact in order to ensure that all generated *ComplianceStates* in the Alloy instance state space are contained in the *ComplianceTrace*.

```
599 fact noOrphanStates {
600   all s: ComplianceState | s in ComplianceTrace·states·elems
601 }
```

6.2 Applying of the Formalization: Printer Example - Access Control

In this last section of this chapter, we introduce the printer example used earlier in Alloy, and model a small regulation for it. The code which is defined here must normally be written by the CoReL and Alloy expert. At a later stage in this thesis (in Chapter 8), we show how the CoReL framework allows to generate this business domain Alloy code, thereby complying with the principles of model-driven software language engineering.

We will start by defining the business model including the specification of all ASEs involved in the compliance modeling. Then we model the system model by defining the SBP. We follow by defining the compliance model, which corresponds to the policies needed. Once we have these steps completed, we have all we need to make Alloy generate the state space, so that we can check its compliance. We do this by defining assertions in the body of the Alloy specification.

First we create a new module for our example:

Module Header

```
602 module printer_ac
603
604 open CoReLexec
605 open sbpexec
606 open util/integer as integer
```

The module is called *printer_ac* and it imports the *sbpexec* and *CoReLexec* as well as the library *util/intereger* modules.

6.2.1 The Business Domain Model

In this section, we will define the various *actions, subjects, entities* needed to model the example using Alloy. We define four different *actions*, two *subjects*, and two different *entities*. The properties of the *subjects* are defined as signature fields. For instance a *user subject* has a *user_type* which is either internal or external.

Defining the ASE Elements of the Business Model

```
607 one sig connect, authenticate, send, print extends Action {
608 }
609
610 one sig user extends Subject{
611   credit_pages: one Int,
612   type : one user_type
613 }{
614   credit_pages = 3
615   type = internal_student
616 }
617
618
619 one sig printer_S extends Subject {
620   realm: one String
621 }{
622   realm = "university"
623 }
624 one sig printer_E extends Entity {}
625 one sig file extends Entity{
626   num_pages : one Int
627 }{
628   num_pages = 2
629 }
630 abstract sig user_type{}
631 one sig internal_student extends user_type{}
632 one sig external_student extends user_type{}
```

Defining the ASEs

Once we have built the separate business domain elements needed, we can combine them in distinct *ASEs*. We define four *ASEs* in this simplified version of the printer example as a fact by setting the different fields of an *ASE* to the appropriate *action (a), subject (s), entity (e).*

```
633 one sig ase_authenticate, ase_send, ase_connect, ase_print extends ASE{}
634
635 fact{
636   ase_send·a = send
637   ase_send·e = printer_E
638   ase_send·s = user
639   ase_print·a = print
640   ase_print·e = file
641   ase_print·s = printer_S
642   ase_connect·a = connect
643   ase_connect·e = printer_E
644   ase_connect·s = user
645   ase_authenticate·a = authenticate
```

```
  ase_authenticate·e = printer_E
  ase_authenticate·s = user
}
```

6.2.2 The System Model

The last step before moving to the compliance modeling is the modeling of the *SBP*. We do this simply by creating all the required *NodeBlocks* to which we assign *ASEs* using a specific Alloy fact. Then we create the *Blocks* we need. In this simple version of the printer example, we only consider a single *AndBlock* (ab1) containing two different *SeqBlocks* (sb1 and sb2).

Then we define the sequence blocks by setting the *blockSeq* field of the *SeqBlock*. This is done by setting the mapping between the *Ints* used to index the *blockSeq*. For the *AndBlock*, since there is no order on the two blocks it contains, we set the *blockSet* field thereof to be the union of the two *SeqBlocks*.

SBP

```
one sig printer_bp extends SBP{}
one sig nb1, nb2, nb3, nb4 extends NodeBlock{}
fact {
  (nb2·node)·se = ase_send
  nb4·node·se = ase_print
}
one sig sb1 extends SeqBlock{}{
  blockSeq = (0→nb1)+ (1→nb2)
}
one sig sb2 extends SeqBlock{}{
  blockSeq = (0→nb3)+ (1→nb4)
}
one sig ab1 extends AndBlock {}{
  blockSet = sb1 + sb2
}
```

6.2.3 The Compliance Model

We first define the rules we will use in the *Policy* definition, then we specify the *Policy* constructs.

Rules

We have two rules which inherit from the *Rule* signature. The *pac_rule_cx* models the rule for a *Context* while *pac_rule_ct* models the rule for a *Control*. Each has different field set, but both define how their evaluation is to be computed through a signature fact. For instance the formula $eval.isTrue\ iff\ p.realm =' university'$ for *pac_rule_cx* says that the *Context* is evaluated to *True* if and only if the printer belongs to the university administration realm.

```
one sig pac_rule_cx extends Rule{
  p:printer_S
}{
  eval·isTrue iff p·realm = "university"
}

one sig pac_rule_ct extends Rule{
```

```
672   u:user,
673   f:file
674 }{
675   eval·isTrue iff not (integer/gte[u·credit_pages,f·num_pages] and u·type in internal_student)
676 }
```

Policy

Finally, we can define a simple *Policy*, which has a *Context* and a *Control*. First, the *printer_policy* has one *Context* called *pac_context* and one *Control* called *pac_control*. Each of these two fields is defined as a separate singleton signature with one field set which is the *rule*. For the latter, we use the *Rules* defined above.

```
677 one sig printer_policy extends Policy{}{
678   pac_context = this·@context
679   pac_control = this·@control
680 }
681 one sig pac_context extends Constraint{}{
682   this·@rules = (0→pac_rule_cx)
683 }
684 one sig pac_control extends Constraint{}{
685   this·@rules = (0→pac_rule_ct)
686 }
687 fun printer_policy::decision: PolicyDecision{
688   (this·@context·holds and this·control·holds) ⇒ resume
689   else interrupt
690 }
```

Chapter 7

CoReL Concrete Syntax

This chapter shows how the CoReL framework is supported by a tool suite built using model-driven technologies. We also show some of the various editors implemented as part of the framework reported on in this thesis.

7.1 The MaRCo Tool Suite

In order to support our CoReL framework with appropriate tooling, we have built a purely model-driven tool suite in our lab. The implementation of this tool suite was a collaborative effort and not solely the work of the author of this thesis[1]. The tool suite is based on the Eclipse framework. It is named MaRCo tool suite after the FNR[2] funded project in the context of which this thesis has been written. In Figure 7.1 we see the various components of the modeling layer of the MaRCo tool suite.

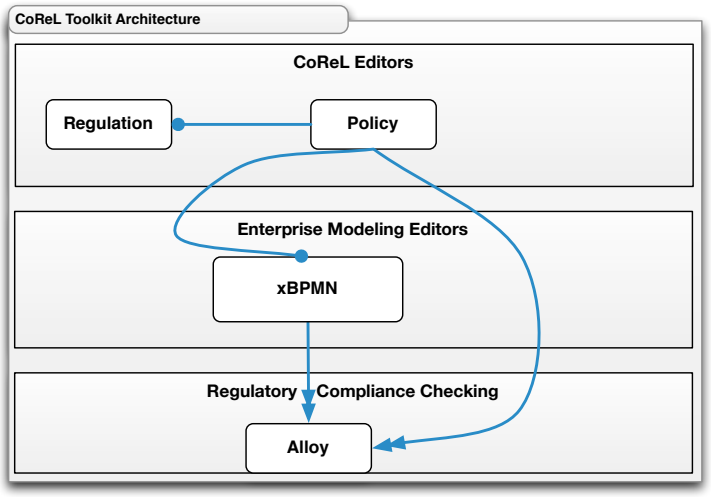


Figure 7.1: The Editor Architecture of the MaRCo Tool

In the MaRCo tool suite, we have three different layers: (i) CoReL policy editors, (ii) Enterprise Modeling Editor, (iii) Regulatory Compliance Checking layer. There are several CoReL editors, the first of which is the Regulation editor, which allows to create regulation models. Then we have the core editor of the MaRCo tool suite, the CoReL policy editor, where policies

[1]Acknowledgements to Christian Glodt who implemented great parts of the comprehensive tool suite. His excellent expertise in various MDE and Eclipse technologies in the production of this tool suite made timely development of the tool possible.

[2]Fond National de la Recherche du Luxembourg. www.fnr.lu

declared in the regulation model are defined. Also CoReL policies might be complex and require to be modeled as a workflow (decision flow editor) or as complex policies using algebraic operators.

We have a second layer of modeling editors, the enterprise modeling editors. We group them in two categories, the first one is the E^{3}PC business process modeling editor, and the second one is the set of modeling editors for enterprise business aspects. For instance, one of such editors is the organizational modeling editor. This editor allow to model organizational entities and relations among these, which are not included in the business process since not immediately relevant for process execution. Users have the possibility to create such models separately from E^{3}PC as these belong to a different domain. The information stored in such enterprise business aspect models is relevant for the enforcement of policies on E^{3}PC business processes.

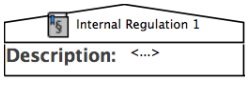
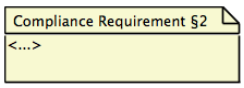
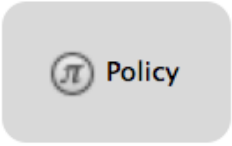
The last layer of the MaRCo tool suite is the compliance checking layer; where we use verification methods: (i) declarative formal modeling using small-scope hypothesis bounded model-checking of Alloy, (ii) Algebraic Petri Net model checking using the model-driven Alpina tool. Finally, in our hypothesis of work, it is possible to use third-party model-checkers provided the adequate model transformation for feeding them the right input is implemented.

7.2 CoReL Symbols

In this section, we present the concrete syntax of CoReL. We proceed by first introducing a summary of all the symbols used for concepts and the relations between concepts. Then we describe the different diagrams used in the language.

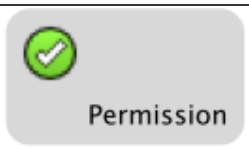
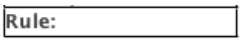
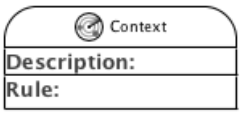
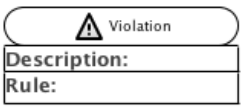
7.2.1 Symbols Table

Table 7.1: CoReL Concrete Syntax Elements

Concept	Symbol	Description
Regulation		The regulation symbol. A regulation may contain or be contained in other regulations.
Compliance Requirement		Compliance requirement symbol. It contains a description of the compliance requirement and a name. It also has an association to a Regulation.
Policy		The policy object (graphical shape) represents a business policy. It is the conceptual encapsulation of a decision-making unit. In order to be enforceable, a policy should specify a deontic modality.

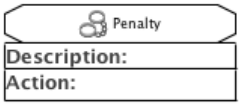
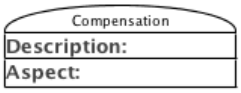
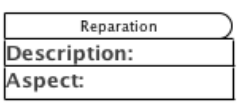
Continued on Next Page...

Table 7.1 – Continued

Concept	Symbol	Description
Obligation		The obligation shape represents a policy with an obligation deontic modality.
Permission		The permission shape represents a policy with an permission deontic modality.
Prohibition		The prohibition shape represents a policy with an prohibition deontic modality.
Rule		The rule compartment can be found in several CoReL shapes and contains a textual rule statement.
Context		The context shape contains a textual description of the context definition for documentation purposes. It also contains a logical rule statement that allows to evaluate when the context holds.
Control		The control shape contains a textual description of the constraint described by the control for documentation purposes. It also contains a logical rule statement that allows to evaluate when the control is complied with or not.
Violation		A violation shape contains a textual description of the violation as well as a rule statement that allows to decide whether the violation occurs or not. The rationale behind this is that we may have several different violations defined for the same policy. We do not only consider the trivial violation that does occur when the control is evaluated to false.

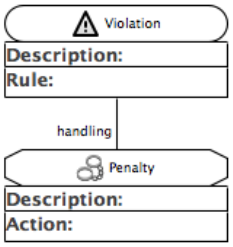
Continued on Next Page...

Table 7.1 – Continued

Concept	Symbol	Description
Reward		The reward shape contains a textual description of the intention behind the reward as well as an action statement to be executed in order to implement the reward. The action description is given in a pseudo-action language for illustration purposes throughout the examples in this thesis. The reward shape is always linked to at least a violation.
Penalty		The penalty shape contains a textual description of the intention behind the penalty (i.e., sanction) as well as an action statement to be executed in order to implement the penalty. As for the other type of handling (i.e., reward), the action description is given in a pseudo-action language for illustration purposes throughout the examples in this thesis. The penalty shape is always linked to at least a violation.
Compensation		As for all other shapes, a first compartment contains the textual description of the compensation, while the second one contains an action to be executed. This action is not relevant for the system model, as it happens outside its boundaries. In our examples we do not use a concrete action modeling language, as it is outside the scope of our research. Instead we use pseudo-code or pseudo-predicates for describing actions.
Reparation		A first compartment contains the textual description of the reparation, while the second one contains an aspect which can be weaved into a system model, such as a business process model. The aspect represents the change that has to be conducted in order to implement the reparation recovery. In our examples we do not use a concrete aspect modeling language, as it is outside the scope of our research. Instead we use pseudo-code for aspects.

Continued on Next Page...

Table 7.1 – Continued

Concept	Symbol	Description
Connection		All connections that represent relations between concepts as defined in the metamodel are displayed as lines. The type of the relation is attached to the graphical connection. In the example shown to the left, the relation between the violation and the penalty is shown as a connection carrying the 'handling' type.

7.3 CoReL Diagrams

The CoReL modeling language encompasses several types of diagrams: (i) Regulation, (ii) policy and (iii) violation diagrams. Regulation diagrams are diagrams showing the breaking-down of regulation in terms of compliance requirements and the link between compliance requirements and CoReL policies. The definition of a CoReL policy is created using a policy diagram, while a violation diagram models the violation decision.

Using these three diagrams we can divide the modeling complexity into three steps. The first step is about allowing the legal expert to proceed to an interpretation of the regulation and extracting as well as structuring the regulatory text in compliance requirements to be modeled.

7.3.1 Regulation Diagram

The regulation diagram is the simplest of all CoReL diagrams. But it is also the only link between the decision-making domain and the regulatory domain. It shows the hierarchical structure of a regulation in terms of compliance requirements as is shown in Figure 7.2. A CR may also contain other CRs.

Thereby, in the case of a regulation model defining a single regulation, we should obtain a tree structure. However, in our work, we do not make the assumption that every CR extracted from a regulatory document is unique, there may be redundancies. One example of such redundant occurrences of a CR in a regulation, are cross-references that may be found inside a regulation. That is one of the reasons why, in the CoReL design, CRs may be shared amongst different regulations (see Figure 7.3).

Another reason for allowing sharing is that we see CRs as the interpretation that an organization makes of elements of an official regulation that applies to its own business. Thus, an organization may decide to interpret two slightly varying textual CRs contained in a regulation as the same in its own context.

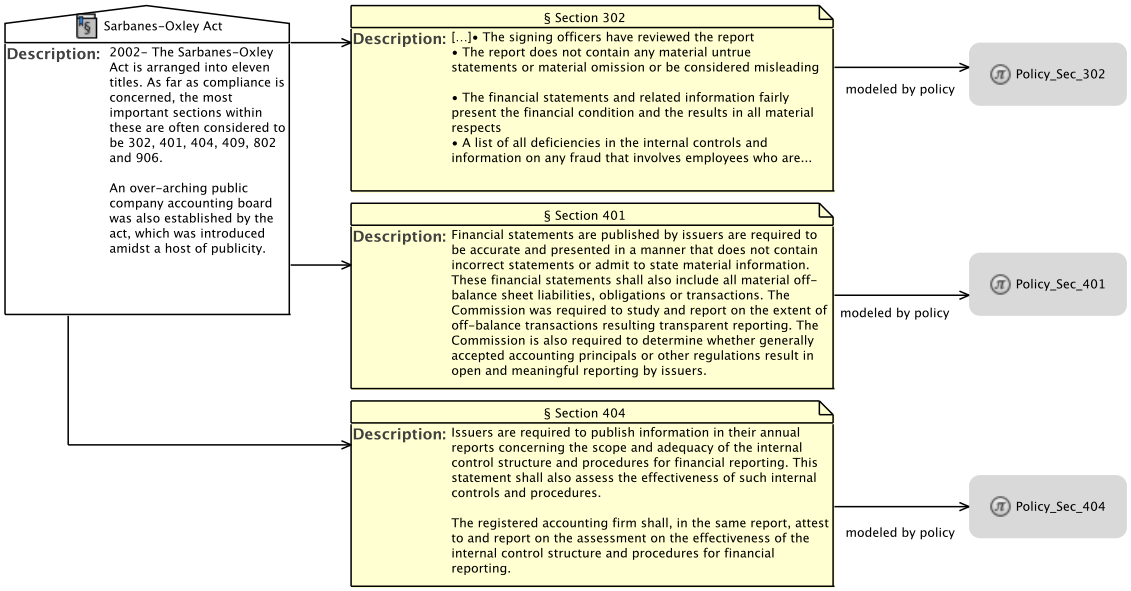


Figure 7.2: Sarbanes Oxley Act 2002 - Regulation Model Excerpt

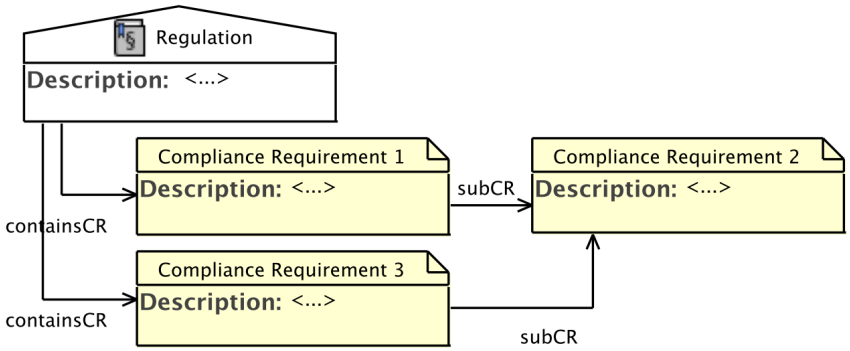


Figure 7.3: CoReL - Multiple Regulations - Compliance Requirement Reuse

Moreover, each compliance requirement which is a leaf in this structure has a unique Policy diagram assigned to, which unambiguously defines its semantics. We introduce Policy diagrams in the next section.

7.3.2 Policy Diagram

The policy expert has knowledge of the CoReL policy language and decision modeling, and accessorily a fair knowledge of rule languages and/or logic. She joins effort with a business expert who has good knowledge of the business processes and the organizational structure of the enterprise. Together they produce CoReL policy diagrams for each compliance requirement.

A Policy diagram is the main diagram of the CoReL language. It gives the definition of a single policy. It shows its context, control, rules as well as possible violations and applying recovery(ies). In the following example we show a CoReL policy diagram from Section 5.6:

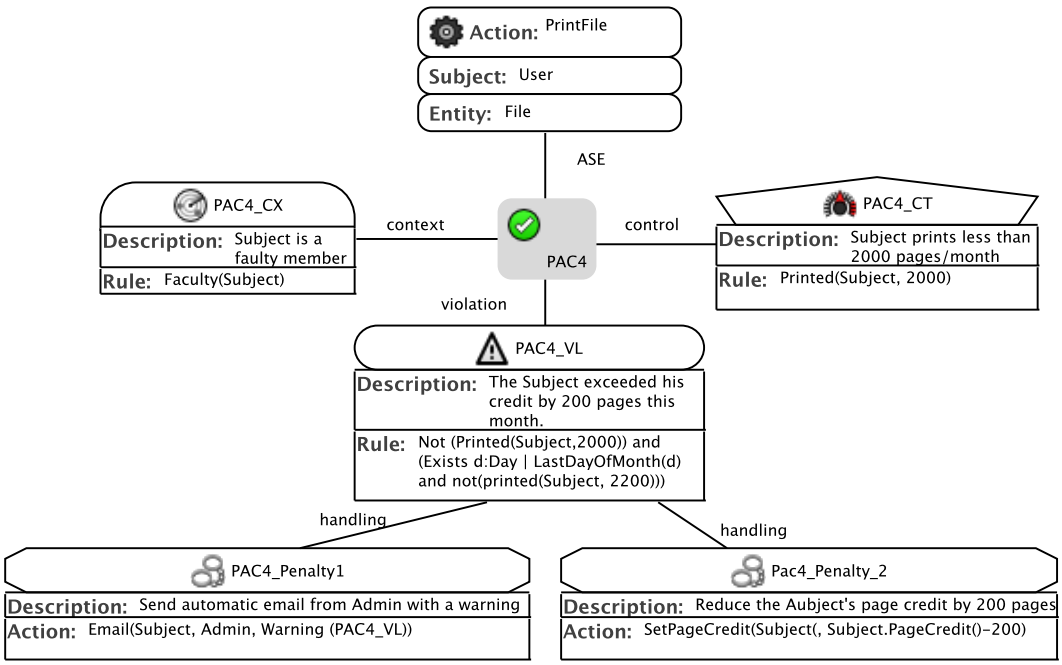


Figure 7.4: CoReL - Printer Example - Random Permission Policy

7.3.3 Violation Diagram

Violation diagrams contain three types of elements: controls, violation values and violations. The connections mapping controls to violation values define a violation valuation function. These connections carry a statement specifying under which condition the targeted violation value is computed. The connections mapping the violation values to a single violation define the violation mapping function. It is also possible to define complex violation valuations by either mapping a set of controls to a violation value, or by mapping a set of violation values to another violation value.

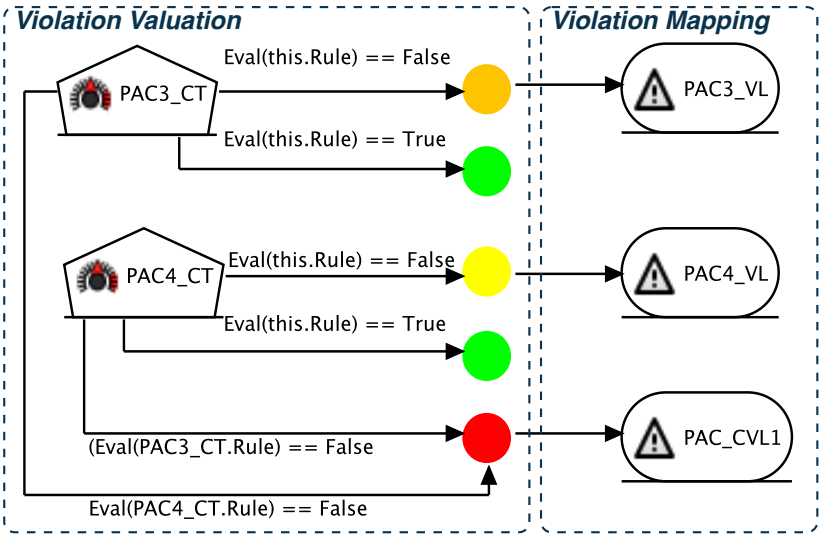


Figure 7.5: Violation Diagram

7.3.4 Remarks on Diagrams

It is noteworthy to say, that the term 'diagram' here should not let the reader ignore the fact that regulation and policy diagrams are in fact formal models. We can consider each diagram as a partial view of an underlying CoReL model. We use the term 'diagram' because it complies with the vocabulary used for the ECLIPSE platform on which our modeling tool is based.

We have now seen two of the CoReL modeling diagrams. These diagrams are the ones linking the textual interpretation of a regulation document, describing its structure, to the set of policies required to model this regulation.

Part III

Regulatory Compliance Verification

Chapter 8

CoReL's Approach to Verification

Long-term commitment to new learning and new philosophy is required of any management that seeks transformation. The timid and the fainthearted, and the people that expect quick results, are doomed to disappointment.

William Edwards Deming.

This chapter explains the verification approach followed in the thesis and guides the reader through the application of MDE principles in the verification of CoReL policies.

8.1 Creating System Models

8.1.1 A DSL for System Models

We need a concrete system modeling language in order to validate the modeling and checking claims using CoReL. We designed a small language named xBPMN for his purpose. In order to be able to quickly produce system models, an EMF editor was implemented in the scope of this research. This editor relies on a slightly different implementation Ecore metamodel from the conceptual one given in Figure 4.8 in Section 4.3, which can express exactly the same workflows as those defined by the SBP language. The xBPMN language can also represent a reduced set of subjects and entities. The metamodel for xBPMN is shown in Figure 8.1.

The models that can be created by this editor are a semantic superset of the SBP language and show one way of how system models may be designed by business users. We chose a syntax inspired by the BPMN [OMG11] language for illustrating how the semantic transition from an established business process modeling notation into our formalization of system models may be realized.

Note that some of the most important modeling constructs of BPMN, namely its rich sub-language of events, has been left out as it is not relevant for our purpose. BPMN events are considered as expert-level constructs and studies have already shown that a very reduced set of BPMN process modelers are likely to use BPMN events in their process models [ZMR08]. It constitutes however an interesting extension of the kind of systems we consider and certainly a challenge for future work.

We chose BPMN because of two main reasons. It offers some possibilities to model operations on resources such as Datastores (e.g., databases, files) or BPMN Messages (can be used to represent general resources being exchanged among tasks). Also, the concepts of Pools and Lanes in BPMN allow to structure the actors involved in carrying tasks and organize the workflow accordingly.

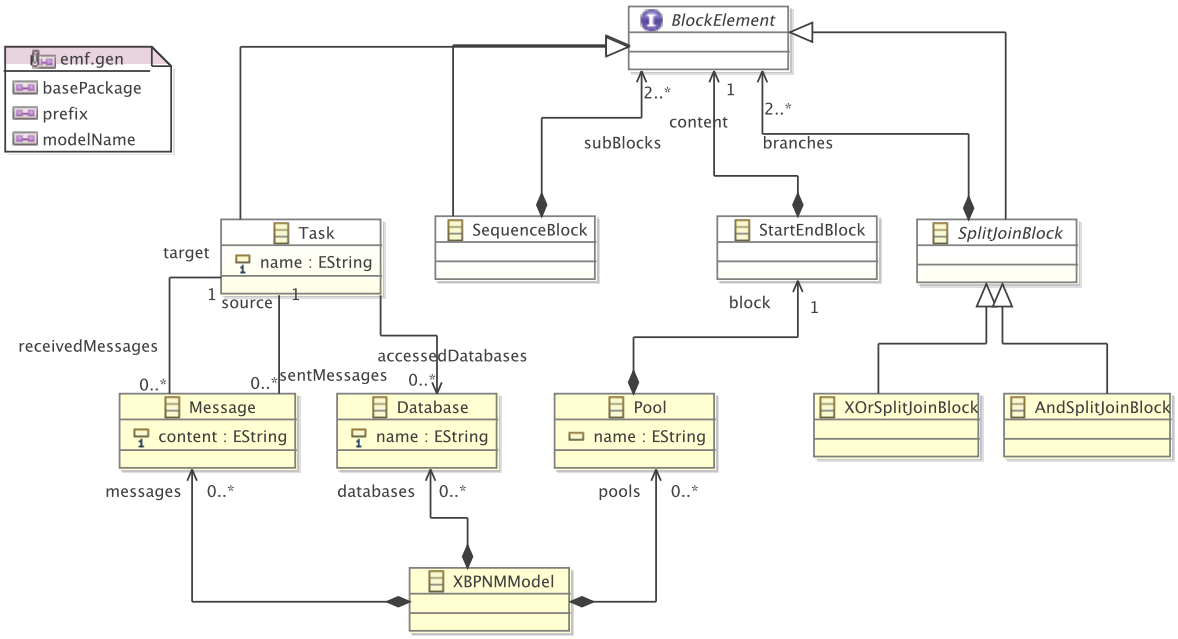


Figure 8.1: xBPMN metamodel in ECORE

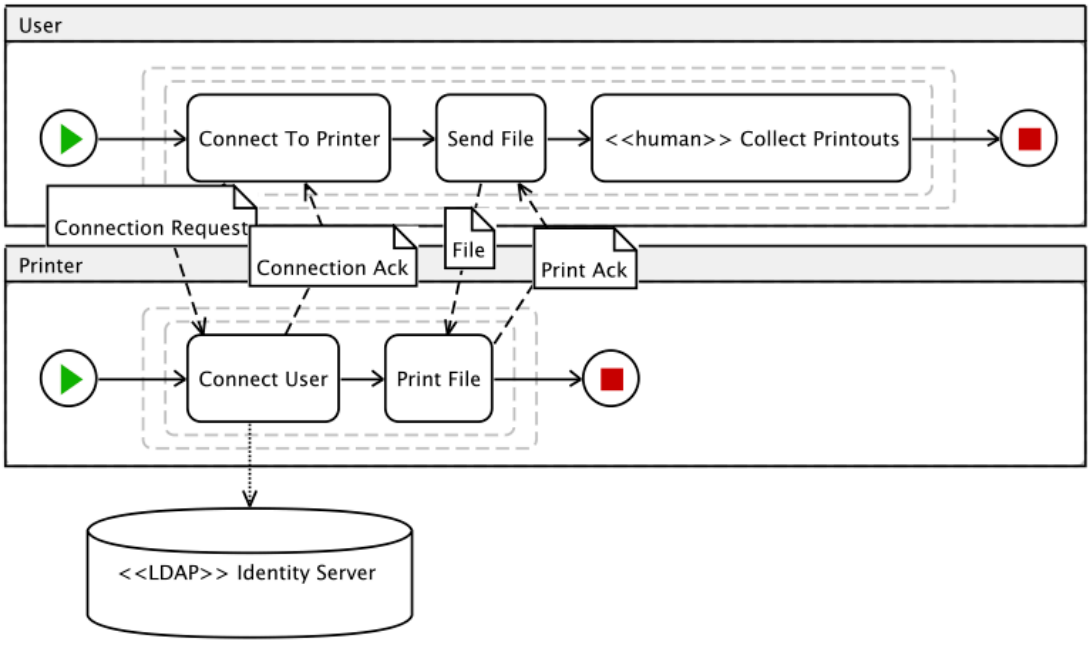


Figure 8.2: The printer Example

Pools and Lanes may be loosely used to represent Roles, Actors, Stakeholders, Responsibles, etc. In xBPMN, we only consider Pools for the sake of simplicity. These latter modeling constructs, allow us to model a reduced set of Subjects and Entites in a workflow, as prescribed by the ASE paradigm. Actions are of course, represented by Tasks. See in Figure 8.2 for an example of how these modeling constructs can be used.

In Section 8.3 on verification of compliance, we show how we use such modeling constructs for generating internal formal representations of a system (i.e. in this case, an xBPMN model).

8.2 Description of the Approach

As said throughout this document, we follow a purely model-driven engineering approach. This implies not only that we develop a formal modeling language but also that we provide model composition and model transformations from this modeling language. In this section, we will show which model transformations we use and why.

8.2.1 Binding CoReL and System Models

First, we must explain how it is possible to use the system and CoReL models jointly. As explained in Chapter 5, we propose CoReL as a semantic bridge between the logic and the business/regulatory domains. In this section, we explain how the concepts of Policy and ASE help bridge the three semantic domains introduced in Section 5.1. This bridging is done thanks to the concepts of Rule, Policy and ASE in CoReL, as illustrated in Figure 8.3.

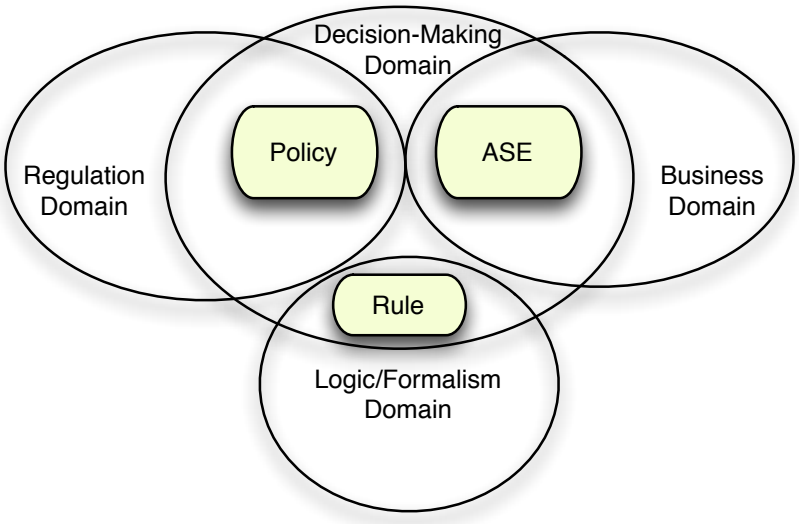


Figure 8.3: Semantic Domain Bridging Through CoReL

The policy concept links the regulatory domain and the decision-making domain where CoReL is based. The ASE concept links the decision-making domain and the business domain, where he enterprise's structure and its processes are described. Similarly, the rule concept links

the decision-making domain and the logic domain.

More specifically, the semantic bridge between the decision-making domain and the business domain is realized through ASE bindings (see Figure 8.4). In order to allow the policy modeler to set values of ASE triples which are valid, a binding is required. This binding is realized at the language (metamodel) level and defines a mapping between ASE triples and elements of a business model.

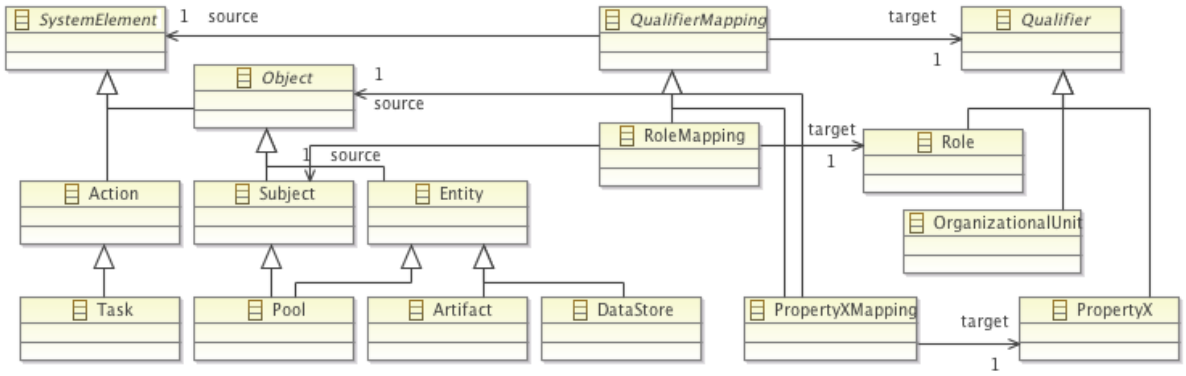


Figure 8.4: ASE Binding at metamodel Level - Example

In the example of binding shown in Figure 8.4, we define a simple language level ASE-binding. We do this simply using inheritance. For the example of xBPMN, we map Process Tasks to Actions, Pools to both Subjects and Entities, and Artifacts (data objects in message flows) as well as Data Stores to Entities. This allows the policy modeler to use xBPMN concepts in the definition of ASEs.

The second step in defining a binding is the definition of Qualifiers. In the example in Figure 8.4 we do this by defining a concept called QualifierMapping which defines a 1-1 mapping between System Elements (i.e., Actions and Objects) and Qualifiers. Both are abstract classes as shown in italic letters in the class name in Ecore.

One example of mapping is the definition of a role for all subjects. Roles are very widely used to define security policies. There are two ways for defining such a mapping. The first way is by re-defining the source and target relations defined for the class *QualifierMapping* for the class *RoleMapping*, source pointing to *Subject* and target pointing to *Role*. The other way does no require re-defining the source and target relations. It is implemented by constraining the classes which the source and target relations of a mapping class (such as *RoleMapping*) point to in the metamodel. If we implement this constraint directly in Ecore we might write them in OCL as in code snippet 8.2.1. The same constraint may be directly implemented in Alloy in case the formalization of the metamodel is done in Alloy.

Code 8.2.1 (RoleMapping Relation Consraint)**.**

context *RoleMapping*
inv: *self·source·oclIsKindOf(Subject)* ***and*** *self·target·oclIsKindOf(Role)*

8.2.2 Alloy-based Verification

CoReL has been equipped with a description of semantics expressed in Alloy, as such Alloy constitutes the semantic domain for CoReL. This makes Alloy an obvious candidate for verifying CoReL models on system models. Since we also provided a description of the semantics of system models, as combinations of ASE and SBP models, in Alloy, it makes the Alloy semantic domain the common integration formalism for both system and CoReL models.

Figure 8.5 shows the overall verification approach of the models used in the verification approach. First, we define model transformations to generate Alloy descriptions of each of the four mentioned types of models: system (i.e., in our case illustrated using xBPMN), regulation, policy and violation models. This maps the business semantic domain to the Alloy semantic domain. This generates an application-specific Alloy module.

The bottom part of the figure shows the Alloy module dependencies according to the use of module imports (i.e., using the *open* keyword). The generated Alloy module must import the *CoReLexec* module and possibly any set of libraries from the *util* library. In the next section, we define the described model transformations.

8.3 Model Transformations

8.3.1 SBP to Alloy

In this section, we will introduce the model transformation rules we define for implementing the SBP to Alloy model transformation. We specify a set of numbered transformation rules given in Figure 8.7 to implement this model to text transformation. The pseudo-syntax uses two character fonts, one in regular police (not bold) is used for transformation rule statements. The other font is bold and in a different police from the transformation rule statements. The second font is used for generating Alloy code, and such Alloy statements are always between two hypens ("<**Alloy Code**>"). The other model transformations are presented using the same convention. Rules follow the definition underneath:

Definition 8.3.1 (Model to Text Transformation Rule)**.**
Rule Number : Matched Pattern (in Model) $\mapsto$ *Applied Pattern.*

Each rule has a matching pattern (on the left side of Figure 8.7) which is what is matched by the rule engine on the model instance to be transformed. On the right side, we find the applied pattern, i.e., what is generated by the model transformation. The model transformation has been defined here in a pseudo transformation language and has been implemented using the *Freemarker*[1] template language which is based on Java. Template-based transformation languages are very often used for Model to Text transformations, because they easily support recursiveness and especially because they use place holders embedded in generated code, which can then be later filled with new code produced by other function calls. Other alternatives could be languages such as JET[2], Xpand[3], Acceleo[4]. The choice of the language here is purely a

[1] freemarker.sourceforge.net
[2] www.eclipse.org/modeling/m2t/?project=jet
[3] wiki.eclipse.org/Xpand
[4] www.acceleo.org

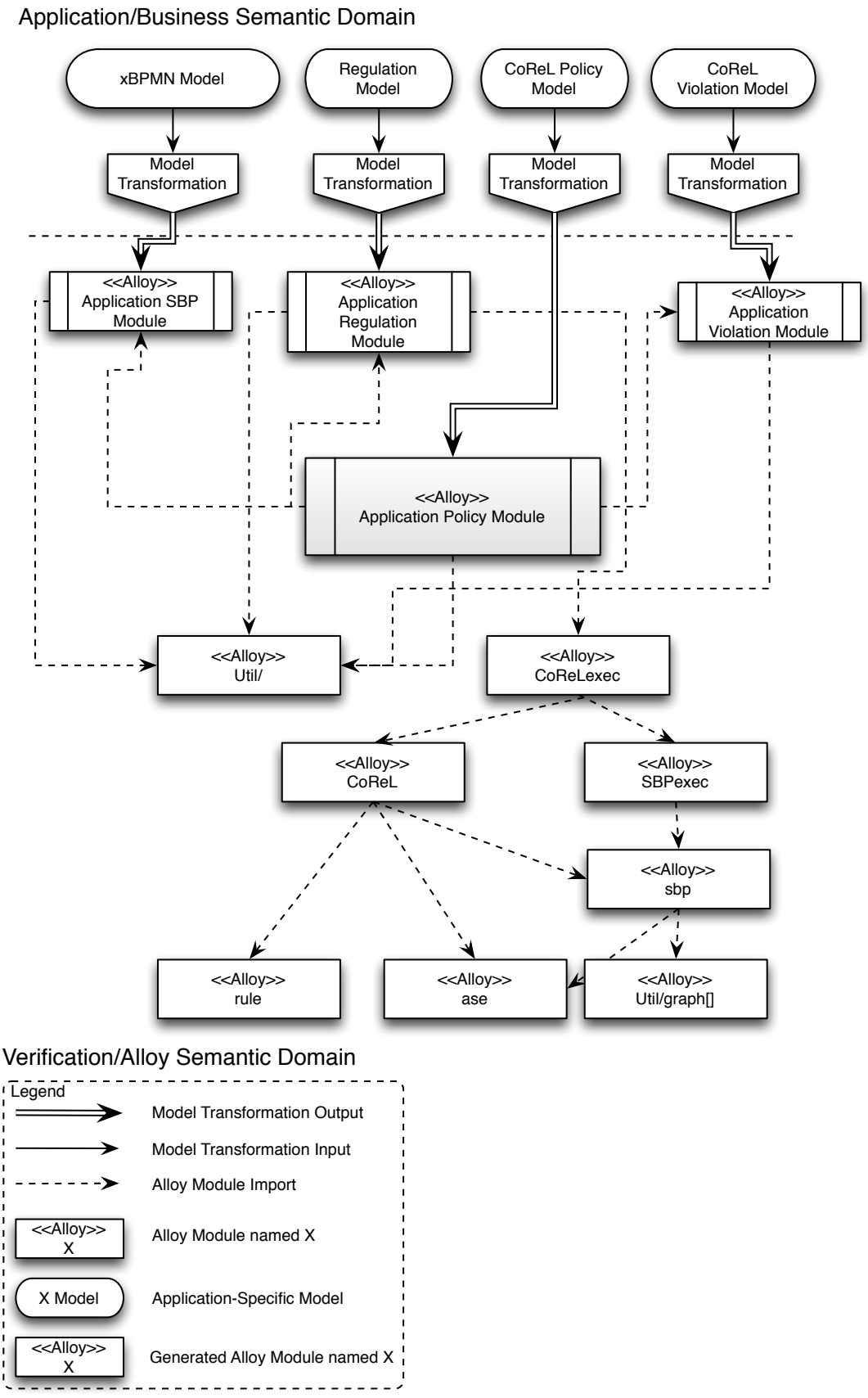


Figure 8.5: CoReL's Approach to Design-Time Verification Using Alloy

meter of personal preference of the researchers, so results can be reproduced with other languages.

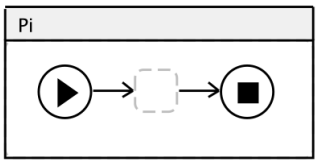
In the following paragraphs, we will explain how each rule is to be understood. In order to make sure the syntax of the rules is understood, we advise the reader to carefully revise the metamodel for xBPMN earlier in the chapter in Figure 8.1. This is necessary to understand (using class and relation names) the parsing and iterating through metamodel classes and relations. Symmetrically, the reader may feel necessary to have a look at the Alloy code describing SBP semantics in Chapter 4. This may be also necessary for the other model transformations we introduce in this chapter.

Tasks

For every process task T_i we will generate three unary signatures. First, the T_i_nb signature defines a single NodeBlock, while T_i_ase defines the ASE which will be put inside that NodeBlock. Then, we finally define the T_i_a Action corresponding to T_i.

In a second phase, we must declare a fact which assigns the right ASE T_i_ase to the T_i_nb NodeBlock. Following this, the rule may set the Action and Subject of the NodeBlock T_i_nb. The Subject of T_i_nb is o course the process Pool variable. The right Subject name will be set when the transformation rule mapping Pools fires (see the rule on the Pool pattern).

Pools



The pools are containers for the tasks executed by different stakeholders. As this information is stored in SBP subjects, we do not need to keep Pools. The first pattern matches only if we have several pools and produces an AndBlock containing the respective workflows in the Pool, each in a respective branch. The semantics of this pattern are illustrated in Figure 8.6. We generate the same Alloy code we would have generated from the AndBlock on the right part of the picture. After this rule fires, the $T_i.Pool() variable is set to the signature P_i.

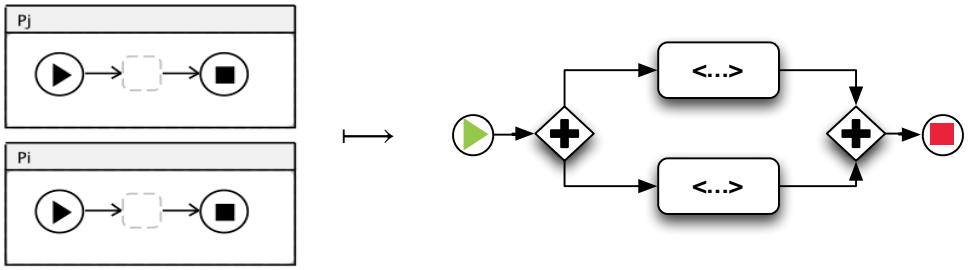


Figure 8.6: Multiple Pools Rule

```
L1: 1. if #Pools > 1          ⟼ "one sig ab extends AndBlock {}{
L2:                                 this.@blockSeq = 0 -> "Pool[0].StartEndBlock" + … + k -> "Pool[k].StartEndBlock"
L3:                             }"

L1: 2. ∀ $T_i in Task         ⟼ "one sig "$T_i.name"_nb extends NodeBlock{}
L2:                               one sig "$T_i.name"_ase extends ASE{}
L3:                               one sig "$T_i.name" extends Action{}"

L1: 3. ∀ $T_i in Task         ⟼ "fact{
L2:                               ("$T_i.name"_nb.node).se = "$T_i.name"_ase
L3:                               "$T_i.name"_ase.a = "$T_i.name"_a
L4:                               "$T_i.name"_ase.s = "$T_i.Pool().name"
L5:                               }"

L1: 4. ∀ $P_i in Pool         ⟼ "one sig "$P_i.name" extends Subject{}"

L1: 5. ∀ $B_i in BlockElement:
L2: Switch ($B_i){
L3:   case: $B_i in SequenceBlock      ⟼ "one sig "$B_i.name"_sb extends SeqBlock{}
L4:                                       {blockSet = 0 -> "$B_i.name"_sb.subBlocks[0] + … + "$B_i.name"_sb.subBlocks[k]}"
L5:   case: $B_i in AndSplitJoinBlock  ⟼ "one sig "$B_i.name"_ab extends AndBlock{}
L6:                                       {blockSet = 0 -> "$B_i.name"_ab.branches[0] + … + "$B_i.name"_ab.branches[k]}"
L7:   case: $B_i in XorSplitJoinBlock  ⟼ "one sig "$B_i.name"_xb extends XorBlock{}
L8:                                       {blockSet = 0 -> "$B_i.name"_xb.branches[0] + … + "$B_i.name"_xb.branches[k]}"
L9: }

L1: 6. ∀ $M_i in Message      ⟼ "one sig "$M_i.name" extends Entity{}{"
L2:                               (∃ $T_k | $M_i.source() == $T_k) ==> "T_k_ase.e = "$M_i.name"_e"
L3:                               (∃ $T_j | $M_i.target() == $T_j) ==> "T_j_ase.e = "$M_i.name"_e"
L4:                               "}"

L5: 7. ∀ $D_i in Datastore    ⟼ "one sig "$D_i.name" extends Entity{}{"
L6:                               (∃ $T_i | $T_i.accessedDatabases().contains($D_i))
L7:                               ==> $T_i.name"_ase.e = "$D_i.name"
L8:                               "}"
```

Figure 8.7: SBP to Alloy Transformation Pattern Rules

Blocks

Depending on the type of Block we match in the xBPMN pool, we apply a different pattern. SequenceBlocks are mapped to SeqBlocks (see SBP semantics in Section 4.4), and the other two types of blocks similarly. Then we set the blockSeq relation of each block. The only type of Blocks where this is crucial are SeqBlocks since a SeqBlock actually describes the order of execution of the ASEs. In the case of the AndBlock and XorBlock, it is sufficient to put all the branches of each block into the blockSeq sequence in any given order. Moreover, according to the formal Alloy SBP module, it is sufficient to define SeqBlocks, as setBlocks are directly derived from SeqBlocks through flattening. Figure 8.8 gives an example for each type of Block.

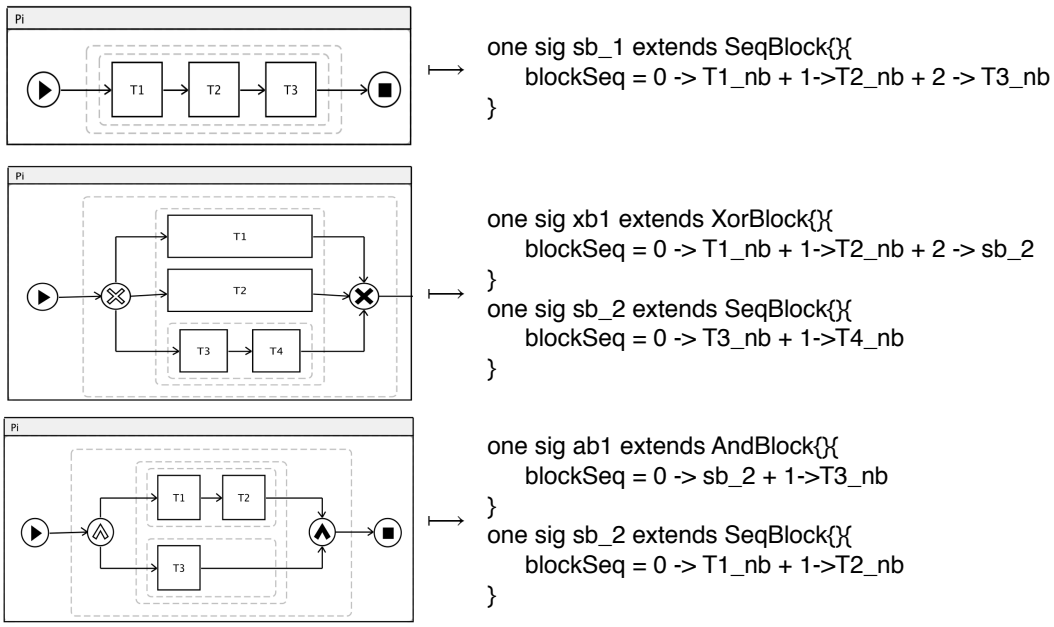
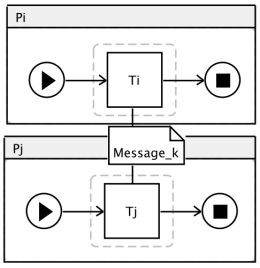


Figure 8.8: Mapping Blocks - Examples

Message



Finally, we get to the two types of Entities we can model in xBPMN. First, messages are sent between tasks in diferent pools. Messages in xBPMN are inspired from a special type of BPMN artifacts called *Data Objects* [OMG11].

We can view a message exchange as two different ASEs happening concurrently and sharing the same entity: the first ASE is about one task sending the message, and the second ASE is about the second task receiving the message. The trick in defining the right applied pattern is just to set the message as the Entity in the right ASE. This is done by accessing both the ASEs where the Action/Task is the source resp. the target of the message.

As illustrated in Figure 8.9, a task sending a message to another, will cause the generation of two consequent ASEs. The first ASE to execute will be with Action *T1_S* and Entity *Resource* while the second ASE will have Action *T2_R* and the same Entity *Resource*. The extensions

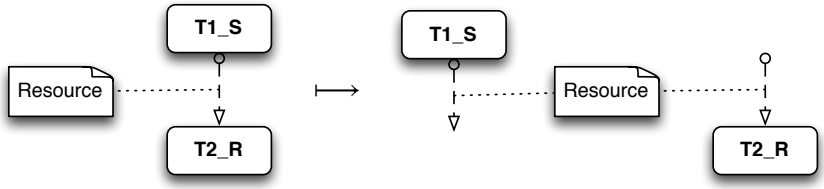
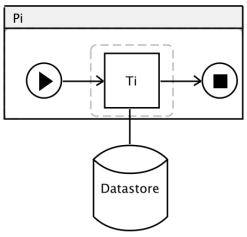


Figure 8.9: Mapping Sending/Receiving Messages

_S and _R used stand for *Send and Receive* respectively. We use an Alloy fact to force the execution of *T1_S* then *T2_R* to follow this order, which we report on in detail in the next chapter in Section 9.2.

For simplification purposes, we do not define complex messaging, i.e., sending and receiving done by the same task, as the order in which the sending and reception must be interpreted are not clear. Therefore, we always assume that a task either sends a message or receives it, but does not do both at the same time.

Datastore



Similarly to messages, we also transform a matched Database (i.e., same use as Datastore in BPMN) access pattern to the Alloy statement which assigns the Database signature as an entity to the right ASE. We do this by accessing the relation that links tasks to Databases. Note that we use the existential quantifier in pattern 7 since we know that a Database can only be accessed by a single task, according to the xBPMN metamodel.

8.3.2 Regulation to Alloy

In this section, we will show how to transform regulation models into corresponding Alloy code. This transformation is aligned with the approach described in 8.2. Regulation models are important in order to know how to map policies and their violations to regulations, or to check the compliance status of a whole regulation. This may be done by defining Alloy predicates which check the current state of all policy atoms (in an Alloy instance model) accessible from a Regulation atom.

The model transformation is shown in Figure 8.10. The transformation is divided in three numbered rules separated by a line, and for each rule, an example of a matching pattern is shown on the left side. Also, a full understanding of the pseudo-rule syntax used in the conceptual presentation of the model transformation can only be achieved by careful understanding of the metamodel for CoReL (see Figure 5.5), since the corresponding class and relation names are used throughout the pattern rules.

The transformation tactics are straightforward and go as follows. In the first pattern rule, for every regulation class in the metamodel we create a unary regulation signature. Additionally,

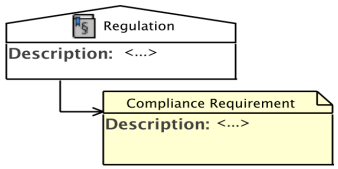
every compliance requirement directly included in the regulation through the metamodel relation *containsCR* is created as a unary signature and is assigned to this regulation using the *complianceReq* relation.

For every compliance requirement which contains no other compliance requirement, called leafCR in the transformation, and therefore may be linked to a single policy, we create a unary compliance requirement signature. Then, in case this compliance requirement has a policy assigned, we use pattern number 2. This leads to the generation of an Alloy code statement setting the *policy* relation in the *ComplianceRequirement* signature.

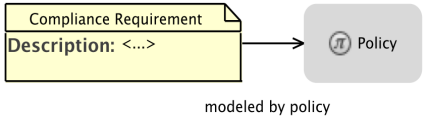
The third pattern rule creates signatures for all the intermediary compliance requirements which are not leaves in the Compliance graph structure (Line 4 in the transformation pattern). The transformation starts by matching all the compliance requirements directly contained in a regulation. For each one, it creates a list of all the indirectly contained sub-compliance requirements. It does so by navigating through the relation named *crContainsCR* in the metamodel. The transitive closure helper function recursively navigates through a given relation and collects all the class instances it encounters until it can not navigate this relation anymore. Such a function is a native operator in Alloy, and has been recently also implemented as part of the OCL language specification [OMG06].

The transformation as we introduce it here deals with structures where we do not have a tree of compliance requirements by not allowing for firing the same rule generating a compliance requirement signature in pattern number 3. We represent this by using the ***Unique*** keyword in line 4 of transformation pattern rule number 3. The statement on line 6 of pattern number 3 sets the subComplianceReq relation in the currently created compliance requirement, in case it contains any sub-compliance requirements.

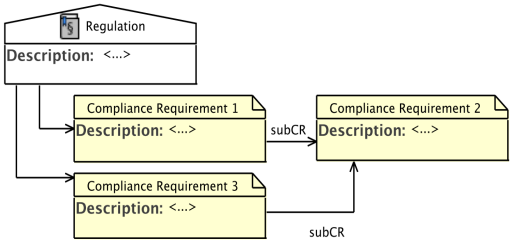
Pattern rule number 4 creates a unary policy signature for each policy concept it encounters. This pattern is an exception in our transformation definition. We do not allow pattern number 4 to be applied. We have included it purely for illustration purposes of how the compliance requirement implementation as a policy would be transformed. The reason for this is that we assume that every policy modeled in a regulation model also has a separate CoReL policy model assigned, and we do not want to declare policy signatures twice in the Alloy code, as this will corrupt the Alloy module (the resulting set of Alloy modules would not be valid). As we will see, it is a lot easier to create that policy signature in the transformation of CoReL policies since we also need to map a policy to its definition elements such as control and context.




```
L1: 1. ∀ $r in Regulation          ⟼ "one sig "$=r.name" extends Regulation{}{
L2:                                      ∀ $cr in $r.containsCR ==>
L3:                                          $cr.name" in this.@complianceReq}
L4:
L5:                                   one sig "$=cr.name" extends ComplianceRequirement{}"
```




```
L1: 2. ∀ $leafCR in ComplianceRequirement | !∃ $cr in ComplianceRequirement | $cr in $leafCR.crContainsCR
L2:                                   ⟼ "one sig "$cr.name" extends ComplianceRequirement {}{
L3:                                        ∀ $p in $leafCR.modeledByPol ==>
L4:                                          this.@policy = 0 -> "$p.name
L5:                                    "}"
```




```
L1: 3. ∀ $cr in ComplianceRequirement | ∃ $r in Regulation | $cr in $r.containsCR
L2:                                   ⟼   ∀ $subCR in $cr.transitiveClosure(crContainsCR) &&
L3:                                          ∃ $cr2 in $subCR.containsCR ==>
L4:                                            Unique("one sig "$subCR.name" extends ComplianceRequirement{}{"
L5:                                                 ∀ $crx in $subCR.containsCR ==>
L6:                                                   $crx.name " in this.@subComplianceReq
L7:                                            }"
L8:                                      )
```

```
L1: 4. ∀ $p in Policy:             ⟼ "one sig "$p.name" extends Policy {}
```

Figure 8.10: Regulation to Alloy Transformation Pattern Rules

8.3.3 CoReL to Alloy

The CoReL to Alloy model transformation is simpler to model compared with the model transformation from xBPMN to SBP (Alloy). Most of the concepts occurring in a CoReL policy diagram are mapped to a single signature. The rest of the concepts include relation definitions and require at least pattern matching rules to be executed.

The model transformation has 6 pattern rules. The first one matches all ASEs in the CoReL policy model and creates a corresponding unary signature. The second and third pattern rules work similarly in that they match and apply similar patterns. Every matched context (resp. control) is mapped to a unary signature context (resp. control). As we already know, context and control are sub-signatures of the same signature constraint. Then, a rule is assigned to each context (resp. control) carrying the same name as the context (resp. control) in the policy model but has an appended extension "_rule_cx" (resp. "_rule_ct") to the name (see line L2 of both patterns).

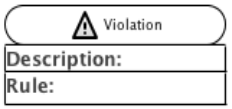
A signature for the rule is also generated in the applied pattern. We implement the truth valuation of the rule by outputting the statement setting the *eval* boolean relation of a rule to be the same as the formula given in the CoReL model. This is shown on line 5 of each of the second (Context) and third patterns (Control). It is noteworthy to indicate that we assume that the rule is written in Alloy for the purpose of this transformation. This statement must be a boolean expression in order for the generated code to be valid. In case the rule statement (i.e., formula) is written in another rule formalism, our Alloy transformation does obviously not work and another kind of semantic domain must be used.
It is not realistic to expect policy modelers to know how to write such statements in the general case. This is why rules are always part of a control or context and must only be modified by an expert. In a practical use scenario, we can imagine a role model for accessing CoReL policy models where only users with an adequate role of logic expert may modify the rules in a Context or Control. However, the intended practice of CoReL modeling is to be able to find and combine available building blocks of CoReL policies (e.g., Contexts and Controls) in order to produce new CoReL policies. This constitutes the main complexity reduction technique used in CoReL and the main advantage compared to existing compliance modeling and checking approaches. Nevertheless, additional tools and techniques such as semantic annotations to CoReL building blocks in order to facilitate retrieval, as well as simulation environments (i.e., for what-if analysis and test scenarios) might make the task of policy validation more successful and the resulting policies more accurate.

The pattern rule number 4 matches any violation in the policy model and generates a unary violation signature. This pattern rule also sets the *recovery* relation of the violation. Pattern rule number 5 creates a unary policy signature and sets all the required relations for complete interpretation of the policy: (i) *governsAse* relating it to an ASE, (ii) *hasContext* resp. *hasControl* relating the policy to its contexts resp. controls, and (iii) *hasViolation* relating to its violations. Finally, the last rule pattern number 6 matches all 4 (there are two types of handling, reward and penalty) types of recoveries. Depending on the type (i.e., penalty, reward, reparation, compensation), rule number 6 produces the required signature declaration.

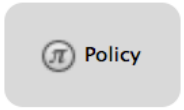
L1: 1. ∀ $a in ASE ⟼ `"one sig "$a.name"_ase extends ASE {}"`

L1: 2. ∀ $x in Context: ⟼
```
"one sig "$x.name" extends Constraint {}{
 this.@rules = 0 -> "$x.name"_rule_cx
}
one sig "$x.name"_rule_cx extends Rule{}{
 eval.isTrue iff " $x.rule.formula
"}"
```

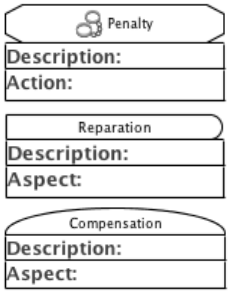
L1: 3. ∀ $t in Control: ⟼
```
"one sig "$t.name" extends Constraint {}{
 this.@rules = 0 -> "$t.name"_rule_ct
}
one sig "$t.name"_rule_ct extends Rule{}{
 eval.isTrue iff " $t.rule.formula
"}"
```



L1: 4. ∀ $v in Violation: ⟼
```
"one sig "$v.name" extends Violation{}{"
    ∀ $r in Recovery ==> $r.name" in this.@recovery
}"
```



L1: 5. ∀ $p in Policy: ⟼
```
"one sig "$p.name" extends Policy{}{"
∀ $a in $p.governsASE ==> $a.name" in this.@governs"
∀ $x in $p.hasContext ==> $a.name" in this.@context"
∀ $t in $p.hasControl ==> $a.name" in this.@control"
∀ $v in $p.hasViolation ==> $a.name" in this.@violation"
```



L1: 6. ∀ $c in Recovery:
L2: Switch ($r){
L3: case: $r in Penalty ⟼ `"one sig "$r.name" extends Penalty {}"`
L4:
L5: case: $r in Reward ⟼ `"one sig "$r.name" extends Reward {}"`
L6:
L7: case: $r in Compensation ⟼ `"one sig "$r.name" extends Compensation {}"`
L8:
L9: case: $r in Reparation ⟼ `"one sig "$r.name" extends Reparation {}"`
L10: }

Figure 8.11: CoReL Policy Model to Alloy Transformation Rules

8.3.4 Violation Decision to Alloy

Violation Models

The violation model is where the mapping between policy controls and violations is described by the policy user. It is shown in Figure 8.12. This is what is called *decision mapping* in CoReL. The first two pattern rules are straightforward, and create unary control and violation signatures respectively for each control and violation in the violation model. We need to ensure in the tool implementation the following modeling constraint: when a control (resp. a violation) used in a given violation model is also defined in a CoReL policy model (as should be the normal case), the first two transformation patterns are not needed (and therefore, shall not be applied) and are given here purely for illustration purposes. If the user does not define (i.e., in a dedicated CoReL model) the control and violation occurring in a violation model, then no violation model can be enforced.

ComputeViolation Alloy Function

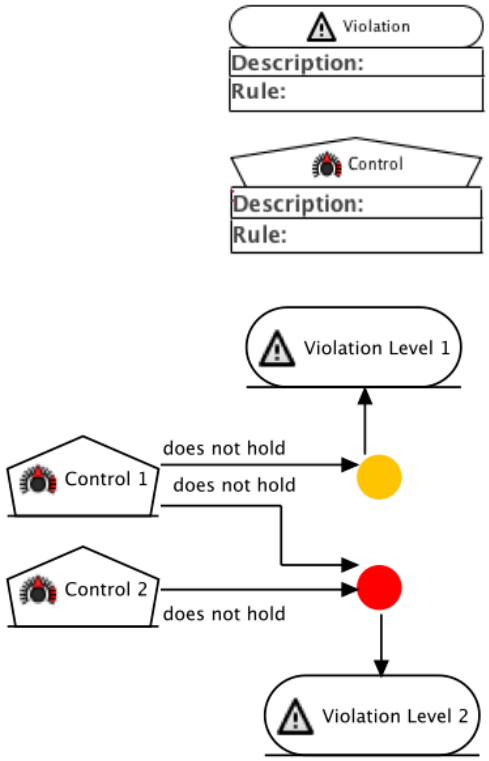
The pattern rule number 3 is the core part of the model transformation. It creates an Alloy function which implements the individual mappings modeled by the user. This function is declared for a policy, which is the policy to which the violation model is assigned. In order to retrieve this information, we must directly implement code into the Eclipse EMF/GMF tool as it is not retrievable using the model transformation language. This is what the statement "*$_Model.AssignedTo.Policy().name*" is for.

The function is called computeViolation. For simplification purposes, we do not append a string such as "*_$policy.name*", in order to uniquely identify the function in case we have multiple policies in a single Alloy module. We assume that we only apply this transformation to a single policy at a time. Otherwise, we would need to extend our implementation to support this. However, this is a practical implementation consideration and it does not reduce the usefulness of our approach.

We must first introduce a set of helper functions we use in this pattern rule before we can go further with the explanation. Helper functions are functions implemented in the transformation language that are to be reused across pattern rules to shorten the pattern rule implementation. We distinguish helper functions from the rest of rule constructs in the syntax as these are displayed using an oblique and underlined font (see Figure 8.12).

LogicalAndJoin Function

We start with the helper function *LogicalAndJoin* used in line 7 of pattern rule number 3. This helper function takes as input a set of boolean conditions and joins them using a logical conjunction in case there are several conditions.



L1: 1.∀ $v in Violation ⟼ **"one sig "**$v.name$**" extends Violation{}"**

L1: 2.∀ $t in Control ⟼ **"one sig "**$t.name$**" extends Control{}"**

```
L1:  3.  ⟼ "fun "$_Model.AssignedTo.Policy().name"::computeViolation : set Violation{
L2:          $y = _Model.violationValues.elems
L3:
L4:          IfElseJoin(
L5:            ∀ i in _Model.violationValues.elems.size() ==> "(("
L6:              ∃ $cv_i in ControlValuation | $cv_i.value == $y[i] ==>
L7:                LogicalAndJoin(
L8:                  $cv_1.completion(EvaluationFormula.serialize()),
L9:                    …,$cv_i.completion(evaluationFormula.serialize())
L10:                   ,…$cv_k.completion(evaluationFormula.serialize())
L11:               )
L12:               " implies "
L13:               ∃! $vd in ViolationDecision | $vd.value == $y[i] ==> $vd.violation.name
L14:         ")"
L15:       )
L16:     "}"
```

Figure 8.12: Violation Model to Alloy Transformation Rules

Valuation Statements and Decision Statement

Before we explain what the second helper function, named *IfElseJoin()*, does, let us first give some definitions. First we define *Valuation Statements*, then *Decision Statements*. A *Valuation Statement* is extracted from the violation valuation, i.e., the set of connections linking a set of controls to a violation value. We define and exemplify it underneath:

Definition 8.3.2 (Valuation Statement)**.**
This statement is a boolean conjunction of boolean conditions on the truth value of a control's rule: (boolean_condition_1 and ... and boolean_condition_k).
Each of these conditions is called Valuation Condition*. They have the form: $control_i$.holds or not $control_i$.holds.*

Example 8.3.1 (Valuation Statement - From Figure 8.13)**.**

Code 8.3.2. *(control1·holds **and** control2·holds)*

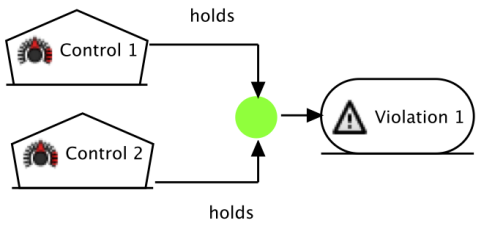


Figure 8.13: Excerpt of a Violation Model

Then we define a *Decision Statement* underneath, and show how it is generated from the example violation model in Figure 8.13.

Definition 8.3.3 (Decision Statements)**.**
A Decision Statement is of the form:
*$DecisionStatement : ValuationStatement$ **implies** $Violation_\alpha$.*
By expanding the $ValuationStatement$ we can write it like this:
*(boolean_condition_1 and ... boolean_condition_k) **implies** $Violation_\alpha$*

Example 8.3.3 (Decision Statement)**.**

Code 8.3.4. *(control1·holds **and** control2·holds) **implies** Violation1*

We allow for types of *evaluationFormulae* to be carried by *Violation Valuation* connections. The first type is the set of two values: {"does hold"; "does not hold"}. The second type is an equivalent set of strings with the identical semantics: {"Eval(this.rule)==True"; "Eval(this.rule)==False"}. We need to transform such *evaluationFormulae* into valid *Valuation Conditions* in Alloy.

Serialize Helper Function

For this purpose, we define another helper function named *serialize()* defined in 8.3.4. This function simply maps the *evaluationFormula* property of the *Violation Valuation* connection (i.e.. "*does hold*" or "*does not hold*") to a valid Alloy boolean *Valuation Condition* (i.e., $Control_\alpha$.*holds* or *not* $Control_\alpha$.*holds*); $Control_\alpha$ being the source of the *Violation Valuation* connection.

Definition 8.3.4 (Serialize Helper Function)**.**
"*doeshold*" *or* "$Eval(this.rule)$ == $True$" $\mapsto$ $Control_{\alpha}.holds$
"*doesnothold*"*or*"$Eval(this.rule)$ == $False$" $\mapsto$ *not* $Control_{\alpha}.holds$

The reader might be puzzled at the redundant ways of expressing *Valuation Conditions* in violation models. We see strings of the form "*does hold/does not hold*" as generic and natural enough to be understood by a large audience of non-computer scientists. The reason why we also allow to include strings of the type "$Eval(this.rule)$ == $True$" and "$Eval(this.rule)$ == $False$" is that it might be preferred by modelers preferring explicit statement of logical conditions. The reader will most probably have intuitively incepted that the *this* in the latter string refers to the $Control_{\alpha}$, the source of the *Violation Valuation* connection.

Initially the "$Eval(this.rule)$ == $True$" and "$Eval(this.rule)$ == $False$" statements were defined in order to allow to write to complex *Valuation Conditions* using several rules in a control, by referring to rules in a sequence of control rules (e.g., written as: "$Eval(this.rule[0])$ == $True$"). As we simplified our language for ease of understanding, we removed that option from the current version of the CoReL tooling.

IfThenElse Helper Function
The third helper function is named *IfElseJoin()* and is used on line 4 of the pattern rule 3. It creates a syntactically valid Alloy *Alternative Choice (i.e., IF THEN ELSE)* statement out of several sub-statements. This helper function is fed as input a list of violation decision statements. The first part of the violation decision mapping statement is the result of applying the helper function *LogicalAndJoin* on a set of *Valuation Conditions*.

Each one of these *Valuation Conditions* is actually carried by the *Violation Valuation* connection (between a control and a violation value) in the violation diagram. The second part in the violation decision mapping statement is the name of the violation that a *Violation Value* maps to.

Definition 8.3.5 (IfElseJoin() Helper Function)**.**
Let i be the number of DecisionStatement in the violation decision statements.
$i = 1$: $IfElseJoin(DecisionStatement_1) \mapsto ViolationDecisionStatement_1$"
$i > 1$: $IfElseJoin(DecisionStatement_1), ..., DecisionStatement_k) \mapsto$
$DecisionStatement_1$ *else* (
$DecisionStatement_2$ *else*
... $DecisionStatement_k$ *else none*
)

Violation Function
We must now define the last aspect in generating the right implementation of the *computeViolation* function. In effect, the violation diagram does not define a complete function over the cartesian product of *Violation Valuations*. We only generate decision statements which include the conditions explicitly modelled by the user. However, to have an unambiguous and complete mapping, we must complete the information extracted from the violation model. In order to illustrate this problem, let us have a look at the Alloy code generated from the example in Figure 8.14.

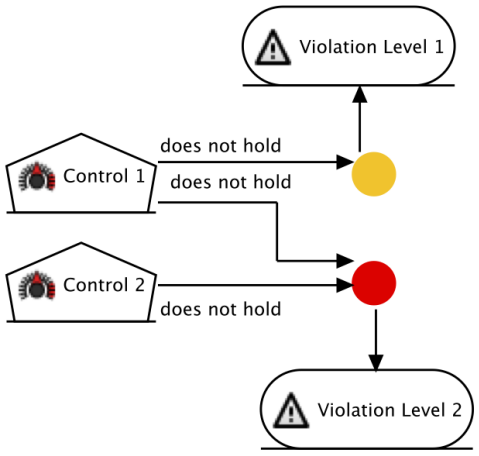


Figure 8.14: Excerpt of a Violation Model

Code 8.3.5 (Excerpt of a Violation Model - Incomplete Mappings)**.**

```
691 fun policy::computeViolation : set Violation{
692   (not Control1·holds and not Control2·holds) implies ViolationLevel2
693    else (
694     (not Control1·holds) implies ViolationLevel1
695       else none
696   )
697 }
```

Finally, the last pattern rule just generates the right violation name to put in the ***implies*** part of every decision statement. It finds the only *ViolationDecision* connection which starts from the *ViolationValue* and traverses its *violation* relation to retrieve the right *Violation* name (see CoReL metamodel in Figure 5.5).

Chapter 9

Validation: Case Studies

In God we trust; all others must bring data.
William Edwards Deming.

In this chapter, we will have a look at some of the modeling features made possible by the ASE, SBP and CoReL formal models. Through the study of some small size examples, we will illustrate the range of modeling capacities made possible by CoReL.

We will also dive a little deeper into the intricacies of our system and CoReL formalization, which are relevant for full grasping of the examples. More concretely, we will see how we model application-specific contexts and controls using rules, as well as recovery modeling and the mathematical description in Alloy of the semantics of violations. Our opinion is that these aspects of the semantics of CoReL are more easily introduced together with an example.

Section 9.1 discusses the range of CRs which are modelled with CoReL. The following sections 9.2, 9.3 et 9.4 present three examples for which we use CoReL to model regulatory compliance requirements.

9.1 Compliance Requirements Dimensions

It is worth saying that, a part of the examples show the decision modeling capacities embedded into CoReL, particularly of flexible decision and violation management. The other part shows the range of rules (i.e., constraints) which can be expressed using our underlying formal model or integrated enterprise (i.e., System as the combination of SBP workflows and ASEs) and decision (i.e., CoReL) modeling.

In our examples, we will cover several types of constraints. We project the constraints along 3 dimensions: (i) static vs. dynamic constraints, (ii) structural vs. temporal constraints and finally (iii) constraints reasoning on violations and meta-policies (relations among policies). We include an explanation of these three dimensions as part of the examples in this chapter. Also, the relevant Alloy code implementing the formal description of the rules is included where needed for better understanding.

We give here a first concise description of these constraints. Static constraints can be checked without executing the process, a classical example being static Segregation of Duties (SoD). These express a standard requirement on executing sensitive tasks by requiring a set of mutually exclusive entities to carry out an action. Dynamic constraints can only be checked when executing the process as they require access to run-time information (e.g., the concrete role

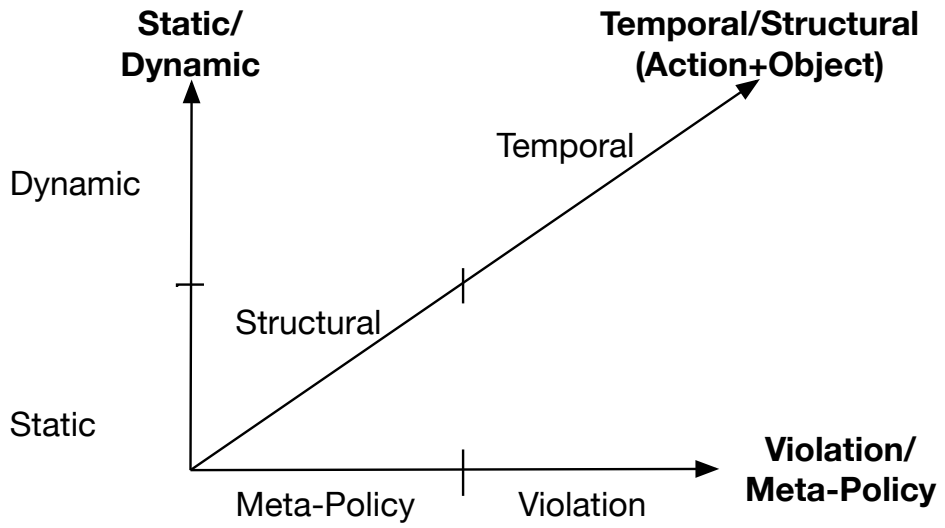


Figure 9.1: The Three Dimensions of Compliance Requirements Modeling in CoReL

of the subject carrying out a given action).

Structural constraints are expressed on the enterprise model elements, i.e., actions and objects, while temporal ones are expressed on the temporal ordering of ASEs in execution traces. Meta-policies describe constraints on the policies themselves, such as conflicting policies (i.e., segregation of policies). Reasoning on violations is also possible, which makes it possible to take policy decisions based on violations which occurred in the past.

The chapter is organized in two separate process examples, each with a set of compliance requirements. After that show how to use CoReL for modeling a process-independent regulation from the healthcare domain.

An important aspect of this work must be clearly grasped by the reader. In this chapter, our method of validation uses examples we refer to as *case studies*. These are self-defined examples, and therefore obviously biased.

In this regard, our validation method is not perfect. It would greatly profit from field studies and carefully designed experiments for validating any claims made about the set of qualities of the artifacts produced in this thesis (i.e., approach, formal model, language). This could be completed by a comparison with other approaches against the same set of qualities. The main claim made in this thesis and which is validated is the feasibility of using CoReL to model an extensive set of compliance requirements. However, this is certainly a highly interesting future track of research on CoReL. Additionally, the examples in this chapter were beneficial to grasp some of the inherent challenges and limitations linked to using CoReL for modeling compliance.

9.2 The Printer Access Control Case

9.2.1 The Compliance Case

This case handles the same simple printer management process we used throughout the thesis to illustrate our research. The process example is shown again in Figure 9.2. We complete the process with a set of compliance requirements shown in Table 9.1 which we explain in detail later in this chapter.

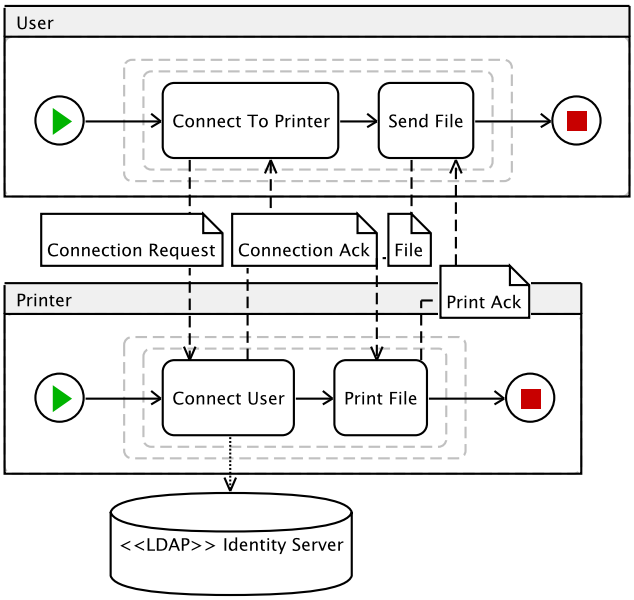


Figure 9.2: The printer Example

First, we propose to carefully study the business process in Figure 9.2. This process has two pools executing concurrently. We have both message exchanges and datastore accesses. These are actions which may be executed by subjects on entities representing classical business process resources. However, our underlying Alloy model for systems does not support tasks sending and receiving messages at the same time. The reason for this is that it is very hard to know from the process design which of the message sending and reception should be executed first.

The same thing goes for datastore access. We do not allow for modeling a task sending/receiving a message while accessing a datastore. An explicit ordering must be specified by the modeler. Our underlying ASE model does require explicit separation of normal process tasks from tasks representing operations on artifacts (i.e., used to model resources).

Therefore, we must transform the process in Figure 9.2 into the process model shown in Figure 9.3. In this process, we show the intended ordering of the process tasks sending or receiving messages, or accessing tasks. The generated Alloy representation is illustrated in Figure 9.4.

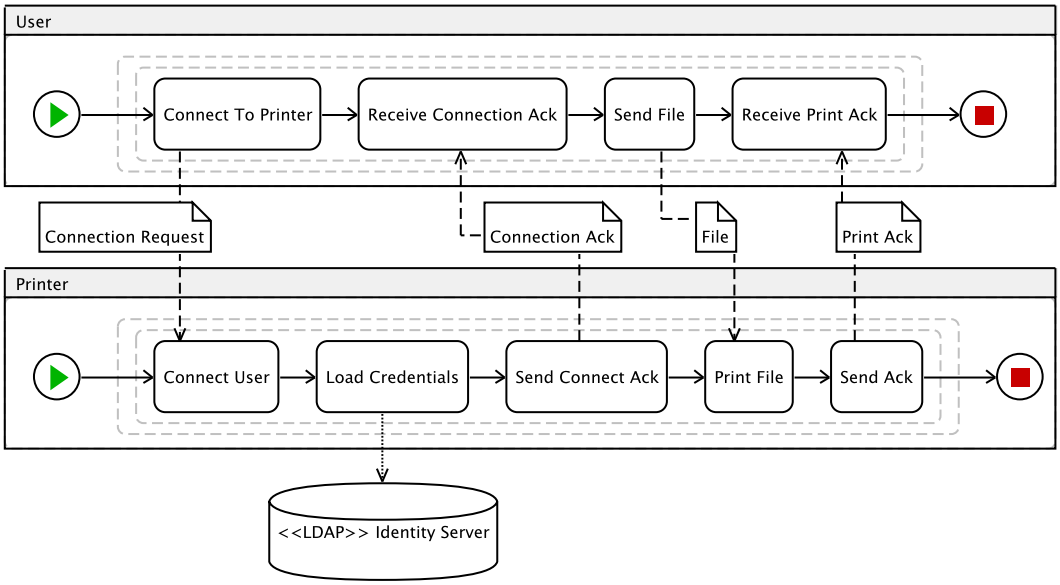


Figure 9.3: The printer Example - Expanded Complex Tasks

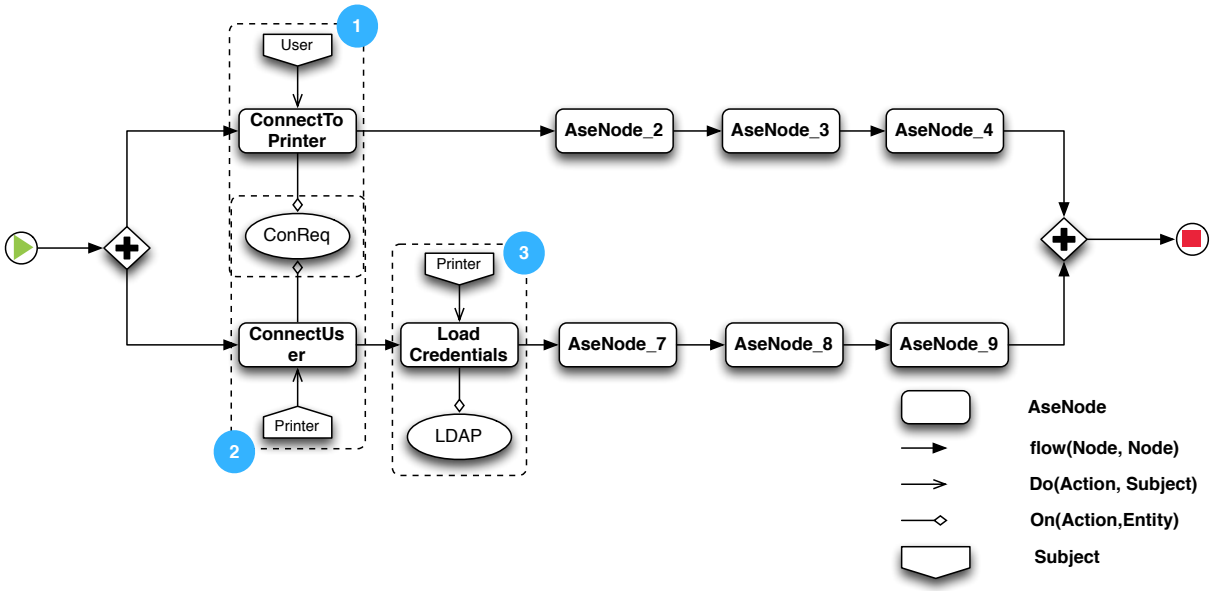


Figure 9.4: The printer Example - Conceptual Illustration of the Corresponding Alloy System Model

For simplification purposes, we do not show the Alloy code, but limit ourselves to the illustration shown in 9.4. The understanding of this Figure should be possible without great effort after reading of the chapter on system modeling 4.3. For ensuring complete understanding, we refer the reader to Section 4.4.1. Rounded rectangles represent actions, pentagons represent subjects, and ovals represent entities. Note that some entities are shared among ASEs.

Also, because of the concurrent execution of branches of AndBlocks, we must explicitly constrain the possible executions of the process model, in order to enforce the ASE execution order implied by the messaging. For instance, we must say that the ASE *(ConnectUser, Printer, Conreq)* must be preceded by the execution of the ASE *(ConnectToprinter, User, ConReq)*. This is done declaraitively in our Alloy formalization by defining a relation *precedes* amongst ASEs.

Code 9.2.1 (Snippet).

```
698 abstract one sig SBP{
699   flow: Node →Node,
700   precedes: ASE→ ASE
701 }
```

After defining this relation, we need to weave its semantics into the execution semantics of SBPs. We do this by adding the following declarative statement to the previously introduced system execution semantics (cf. Section 4.4). We must extend the declarative specification of valid system execution traces accepted by our model. The following Alloy code snippet only shows the added part to the signature fact of a Trace (not to confuse with ComplianceTrace). This statement forces all ASE nodes a in a relation $precedes(Trace, a, b)$ with another ASE b to occur (be into the set $activeSets$) first in the trace before the ASE b.

Code 9.2.2 (Snippet).

```
702 one sig Trace {
703   states: seq ProcessState,
704   activeSets: Int →Node
705 }{
706   all a,b: ASE| a→b in SBP·precedes implies (all j: Int| all n2: AseNode| n2 in activeSets[j] &&
707   n2·se =b ⇒ some i: Int| some n1: AseNode| i< =j && n1 in activeSets[i] && n1·se = a)
708 }
```

9.2.2 The Compliance Requirements

Table 9.1 shows a list of 6 compliance requirements we model in this first example. Each of these CRs is modelled separately in the rest of this section.

9.2.3 CoReL Engineering of the Compliance Requirements

9.2.3.1 CR1

In this first example, the policy decision is about whether or not granting or denying the execution of printer access. The ASE is when a user prints a file. The chosen modality is a permission. The context of the policy is set to hold for students executing the ASE. This means that non-students are not under the jurisdiction of the policy, and another policy is needed if we want to control their access to the printer. We need two controls, one

Table 9.1: Compliance Requirements - Printer Example

CR	Definition
CR1	The file can only be printed by internal students who have sufficient printing credit.
CR2	For the previous CR (CR1), in case the printing credit of the internal student is less than 10% of his monthly (maximal) credit, he must be notified of this situation.
CR3	In case a student does not possess sufficient credit for printing, a message with the explanation of the reason for the refused access to the printer is sent to the user and to the system administrator, and the printing must be denied.
CR4	If the printer is a laser Poster color printer then access is granted only if the user additionally has temporary high-rank clearance.
CR5	If the user has a negative printing credit, then he cannot connect to the printer before he pays a minimum fee of 15$.
CR6	When the User connects to the printer, if the user has only printed less than 50% of his credit for the past month, then an additional 20 pages are added to his credit.

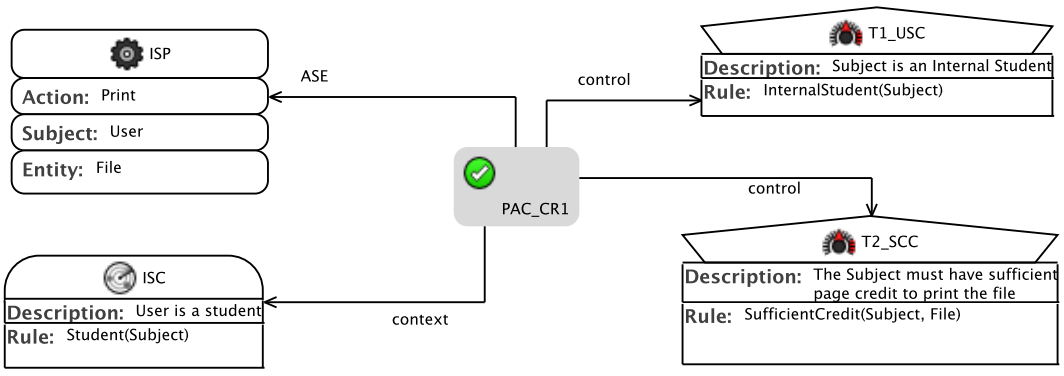


Figure 9.5: CR1 - Policy Model

holding for internal students, and the other holding for a student who has sufficient printing credit.

A small clarification of the semantics is needed here. According to CoReL semantics, when a policy governs a given ASE, it must take an execution decision when this ASE is active during the execution of a system. This decision may be a permission (when all controls hold). However, when there is no policy governing an ASE, that means that no decision needs be made by CoReL about the execution of that ASE, which is equivalent to a permission.

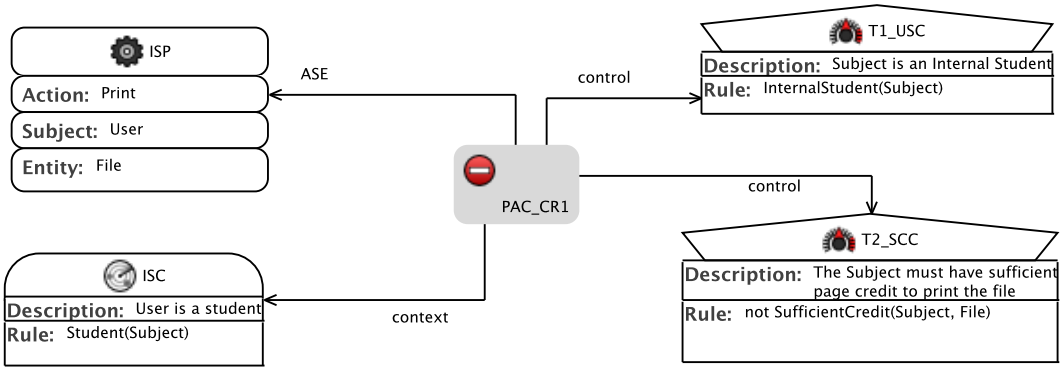


Figure 9.6: CR1 - Alternative Policy Model - Prohibition

In Figure 9.6, we show the alternative way of modeling the CR1. Instead of modeling a permission, we can model a prohibition, by negating one of the control rules and not the other, and leaving the context rule as they are. The reason is that CoReL authorization semantics enforce a modality when all the controls hold. And the desired outcome we would like here is for printing to be denied in case the student is internal but has no sufficient credit. Note that the policy says nothing about what to do when the student is not an internal one.

9.2.3.2 On Application-Specific Rule Implementation & Evaluation

The implementation of the rules we write inside contexts and controls will only be shown in case of non trivial rules (e.g., temporal rules). Otherwise, we only explain the meaning of the rules informally. In order to show how the evaluation of rules of contexts and controls is done in the general case, let us have a look at the Alloy formalization 9.2.4.

This first code snippet shows how we extend the signature of a rule in the separate *Rule* module to include a boolean field *eval*. This way, we can define application-specific implementation of a rule declaratively. The predicate *holds* defined on the rule signature returns $True$ if and only if the boolean *eval* returns $True$.

Code 9.2.3 (Snippet)**.**

```
709 abstract sig Rule{
710   eval: Bool
711 }
712 pred Rule::holds{
713   this.eval.isTrue
714 }
```

The following code snippet shows one context *pac_ context* resp. one control *pac_ control*. Each is implemented using a single rule *T1_ USC_ rule* resp. *T1_ SCC_ rule*. The *user* and *file* signatures used here (e.g., *u:user* in *T1_ SCC_ rule*) are just for illustration purposes in order to be able to evaluate whether rules hold or not. In the normal case we have signatures defining the parameters taken in by a rule (i.e., the predicates used in a rule).

Code 9.2.4 (Snippet).

```
715 -- *******the 1st control *******
716 one sig T1_ USC extends Constraint{}{
717   this·@rules = (0→T1_ USC_ rule)
718 }
719 -- *******the 2nd control *******
720 one sig T1_ SCC extends Constraint{}{
721   this·@rules = (0→T1_ SCC_ rule)
722 }
723 -- *******the 1st rule *******
724 one sig T1_ SCC_ rule extends Rule{
725 }{
726   eval·isTrue iff (integer/gte[user·credit_ pages,file·num_ pages])
727 }
728 -- *******the 2nd rule *******
729 one sig T1_ USC_ rule extends Rule{
730 }{
731   eval·isTrue iff (user·type in internal_ student))
732 }
733 one sig user extends Subject{
734   credit_ pages: Int,
735   type : user_ type
736 }{
737   credit_ pages = Int[3]
738   type = internal_ student
739 }
740 one sig file extends Entity{
741   num_ pages : one Int
742 }{
743   num_ pages = Int[2]
744 }
```

We see that a rule is defined to hold if and only if a given predicate holds (returns *True*). But this will be implemented differently from application to application and also will depend on the checking formalism, which does not need to be Alloy. Therefore we won't further discuss the implementation details of trivial rules. In the rest of this chapter, we will not implement each rule separately. Instead, we will simulate if a rule holds or does not hold by using an Alloy trick shown in the code snippet 9.2.5.

Code 9.2.5 (Snippet).

```
745 one sig dummy_ rule_ holds extends Rule{}{
746   eval·isTrue iff (no none)
747 }
```

```
748
749 one sig dummy_rule_does_not_hold extends Rule{}{
750   eval·isTrue iff (no univ)
751 }
```

The rule *dummy_rule_holds* is always evaluated to True using a tautology trivially holding (i.e., the empty set *none* is empty). On the opposite, the rule *dummy_rule_does_not_hold* never holds as we use a trivially never holding constraint, namely, that the set *univ*, which is the set of all Alloy atoms (the highest super signature in the graph of signatures) is empty.

9.2.3.3 CR2

CR2 extends CR1 by introducing one violation, as shown in Figure 9.7. This violation can only be interpreted when the policy model is accompanied by a violation model. This is shown in Figure 9.8.

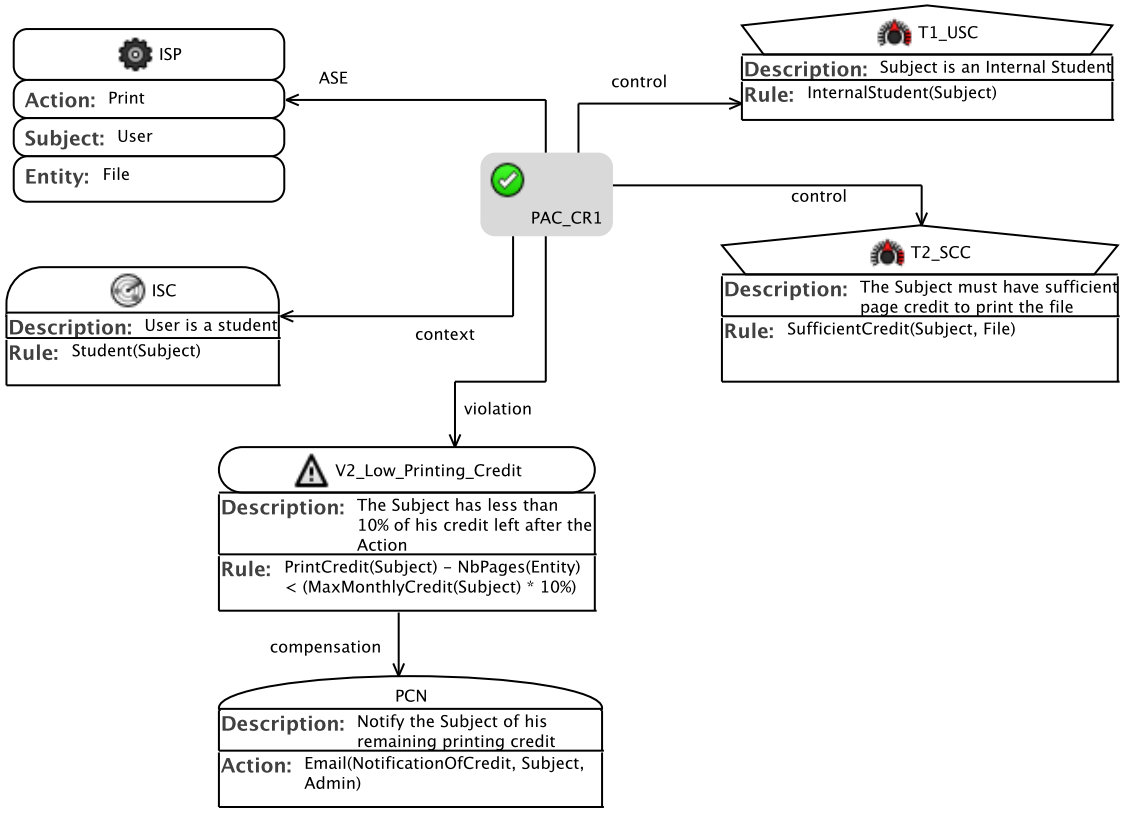


Figure 9.7: CR2 - Policy Model

The violation in Figure 9.8 is only fired when both controls *T1_USC* and *T2_SCC* hold. The violation value which is computed in this case is *Gray*, and this fires violation *V2_Low_Printing_Credit*. In this case, the rule embedded in the violation is evaluated and in case the rule holds, the recovery (compensation) *PCN* prescribed by the violation is enforced. This means that the user will only get a warning message about his low credit in case he was allowed to print.

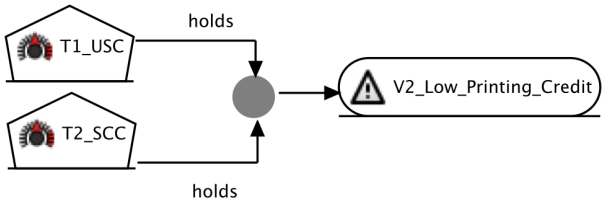


Figure 9.8: CR2 - Violation Model

9.2.3.4 On Application-Specific Violation & Execution Decision

The reader might be wondering about how the policy interpretation engine (as described by CoReL semantics) is able to decide which violations to trigger. We know that this is done using an Alloy function called *computeViolation*. Such a function must be defined for every policy and is generated from the violation model. The function generated for the violation model in Figure 9.8 is given here:

Code 9.2.6 (Snippet)**.**

```
752 fun printer_policy::computeViolation : set Violation{
753   (this·control[0]·holds and this·control[1]·holds) implies this·violation[0]
754   else none)
755 }
```

In the following in snippet 9.2.7, we show how to model recoveries for a violation. We do not distinguish between the different (three) types of recovery in CoReL and show how the generic case of recovery is treated. Let us model the case in CR2, with a single violation and a single recovery, which we name *Violation1* and *Recov1* for more simplicity. First, we must extend the signature *Violation* with a *recoveryRule*, so that we can evaluate violation rules.

Code 9.2.7 (Snippet)**.**

```
756 abstract sig Recovery{
757   rase: ASE
758 }
759 abstract sig Violation{
760   recoveryRule: Constraint,
761   recovery : lone Recovery
762 }{
763   some p:Policy | some i: Int | this in p·violation[i]
764 }
765
766 one sig Recov1 extends Recovery {}
767
768 one sig Violation1 extends Violation{}{
769   recoveryRule = Violation1_rule
770   recovery = Recov1
771 }
772
773 one sig Violation1_rule extends rule{}{
774     eval·isTrue iff ((PrintCredit[Subject] − NbPages[Entity])< (MaxMonthlyCredittenPercent(Subject) ) )
775 }
```

At this stage of explaining the example, the only missing element is how the execution decision of the policy is computed in case of a violation. In CoReL, an execution decision takes either one of two possible values: {*Abort, Resume*}. *Resume* means that the process execution must be continued, while *abort* halts the process execution definitely. The execution decision is enforced only after the violation recovery has been enforced.

Execution decision-making is implemented in CoReL using a combination of extensions to our model. First, we add a boolean relation to the signature *Policy* named *abort*.

Code 9.2.8 (Snippet).

```
776 abstract sig Policy{
777   modality: one Deon,
778   governs: ASE,
779   context: Constraint,
780   control: seq Constraint,
781   violation: seq Violation,
782   abort: Bool
783 }
```

The following step in defining execution decision semantics is to define in which case does the *abort* field become False. In order to achieve this, we must add a fact specifying the latter condition. The condition is based on which violation has been computed by the policy. These are obtained from the violation model.

We have defined the usage principles of *ViolationValues* in order to allow for automatic generation of such statements, and this is where *ViolationValues* come to use. The *ViolationValues* are divided into two subtypes defined by a subset: The *Abort* and the *Resume* subsets. In our proposal of using the *Lights Violation Model* as a *Violation Type*, we have implemented the two subsets as follows: *Abort*={Red, Orange, Yellow} and *Resume*={Green, Blue, Gray}. This design implies that wherever the violation computed is linked to a *ViolationValue* belonging to the *Abort* subtype the decision to *Abort* is taken.

Code 9.2.9 (Snippet).

```
784 fact{
785   isTrue[PAC_CR1·abort] iff (computeViolation = Violation1)
786 }
```

In the code snippet above, we show how the execution is taken for a hypothetical *Violation Violation1* which is linked to one of the *Abort ViolationValues* for the *Policy PAC_CR1*. For the case where we have more than one violation leading to an *Abort* execution decision, we can model it by exploiting the fact that the *computeViolation* Alloy function returns a set of violations like this:

Code 9.2.10 (Snippet).

```
787 fact{
788   isTrue[PAC_CR1·abort] iff (computeViolation in {Violation1 + Violation2})
789 }
```

9.2.3.5 CR3

CR3 (cf. Figure 9.9) adds another violation to the policy and an accompanying violation model (cf. cf. Figure 9.10). The new violation only fires when *T1_USC* holds and *T2_SCC* does not hold. In this case the decision to abort the process will be taken. Before that the violation recovery is executed as the violation carries a trivially true rule.

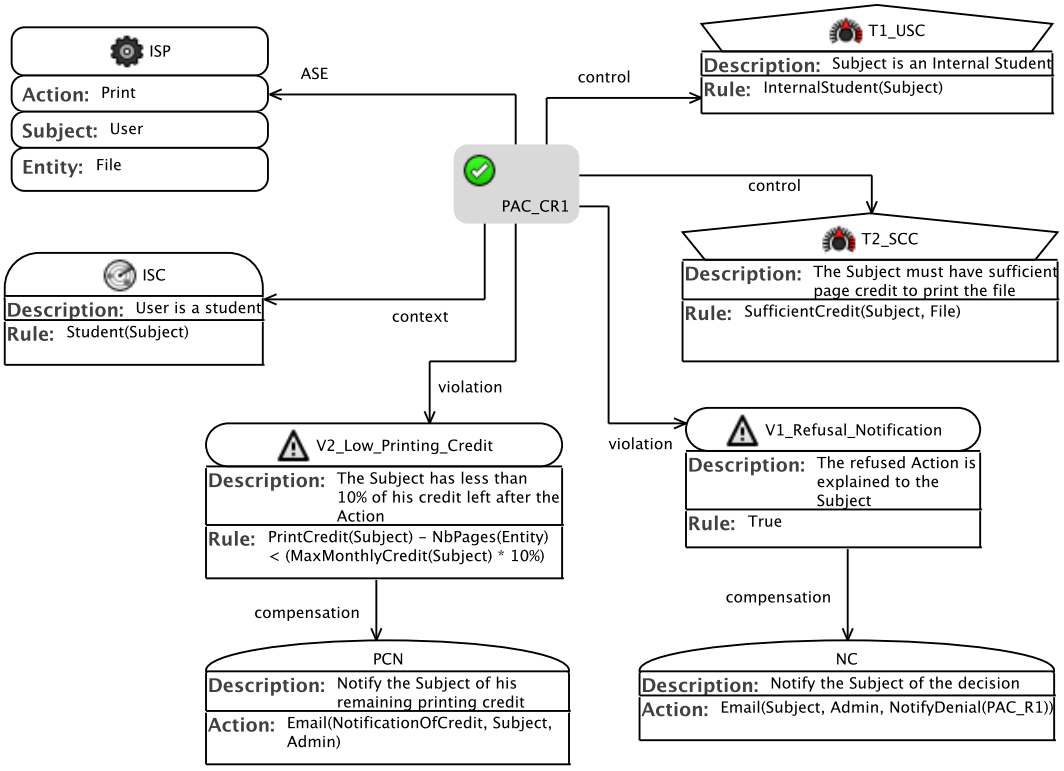


Figure 9.9: CR3 - Policy Model

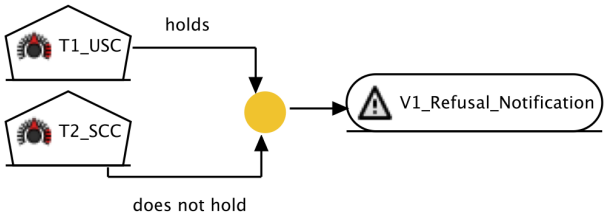


Figure 9.10: CR3 - Violation Model

9.2.3.6 A Violation Model for CR1, CR2 & CR3

We must now group the two separately modelled violation models into a single model for easier visualization (cf. Figure 9.11).

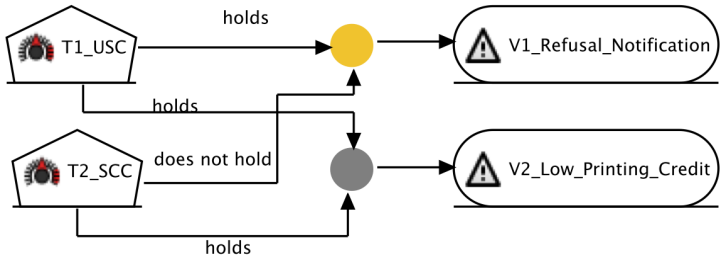


Figure 9.11: Violation Model for CR1, CR2 & CR3

9.2.3.7 Prohibition Version - CoReL Models - CR1, CR2 & CR3

Figure 9.12 shows the equivalent policy model to the one in Figure 9.9. The difference is that the opposite modality to a permission is used, and some of the control rules are negated (only *T1_SCC* in our case). CoReL modelers may in consequence use the modality that they feel corresponds best to their understanding of the CR, knowing that the obtained model will lead to the same enforcement, but modelers must be very careful with the control rules they will use.

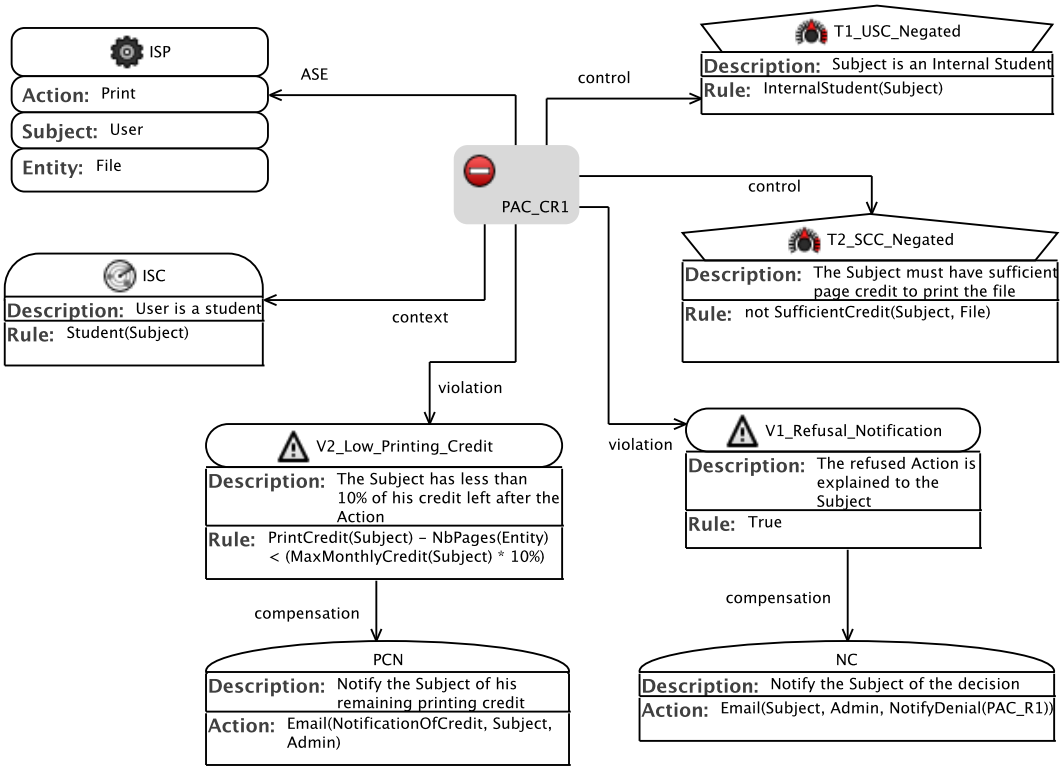


Figure 9.12: CR1/CR2/CR3 - Alternative Policy Model - Prohibition

Also, we show in Figure 9.13 that the violation model that must come with the policy model in Figure 9.12 that the mappings that lead to the two possible *ViolationValues* we consider, *Gray and Yellow* are switched. When changing the modality of the CoReL authorization, modelers must carefully construct the corresponding Violation Models.

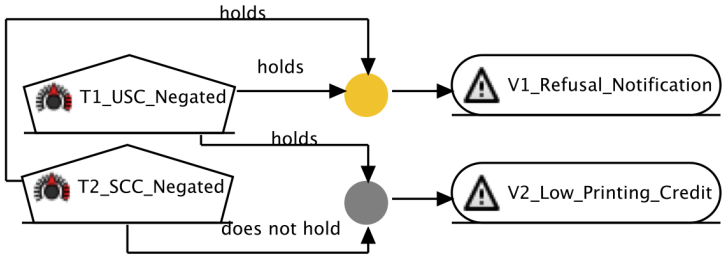


Figure 9.13: CR1/CR2/CR3 - Alternative Violation Model - Prohibition

9.2.3.8 CR4

CR4 is a very interesting compliance requirement, as we find no elegant way to natively model it using the current version of CoReL. The reason for this is that we cannot write rules on ASEs which can access the target of an action. In CR4, we can first try to use one of two ASEs tagged 1 or 2 in Figure 9.4: (ConnectToPrinter, User, ConReq) or (ConnectUser, Printer, ConReq). However, none of these ASEs contains information on both the source (i.e., User) and target (i.e., Printer) of the interaction.

As our ASE model only covers ASEs, we would need to extend it to model action targets. If we replace our ASE triples with quadruples of the form (Action, Subject, Entity, Target), with *Target* being the target of an *Action*, our model will be able to express CRs like CR4.

One could try to model CR4 by using a workaround. We see at least two possible solutions. We could enrich the action signature with an additional field called target. But that is not a satisfying solution since the same action may have several targets, depending on the process it is involved in. Another tentative solution might be to enrich the qualifiers (signatures related to objects carrying additional properties, e.g., role qualifier for subjects) of messages with a destination. But that is not a generic solution and is in fact just hiding the information about an action's target into the message for the specific case where the action is indeed sending a message.

9.2.3.9 CR5

In CR5, modeled through Figures 9.14 and 9.15, we see how to use a new kind of predicate made possible by our underlying formal model. The predicate used in the violation *Provision_ Violation_ CR5* is one example of how a provision, i.e., an obligation which must be fulfilled punctually before execution is resumed or aborted, can be modeled.

The violation *Provision_ Violation_ CR5* is decided upon very simply in the case where the single control does not hold. However, the rule of that violation is explained by showing the corresponding Alloy specification:

Code 9.2.11 (Snippet)**.**

```
790 pred Trace::precedes[a,b: ASE] {
791   all j: Int| all n2: AseNode| n2 in this·activeSets[j] &&
792     n2·se =b ⇒ some i: Int| some n1: AseNode | i< =j && n1 in this·activeSets[i] && n1·se = a
793 }
794
795 pred precedesAlways[a,b: ASE] {
```

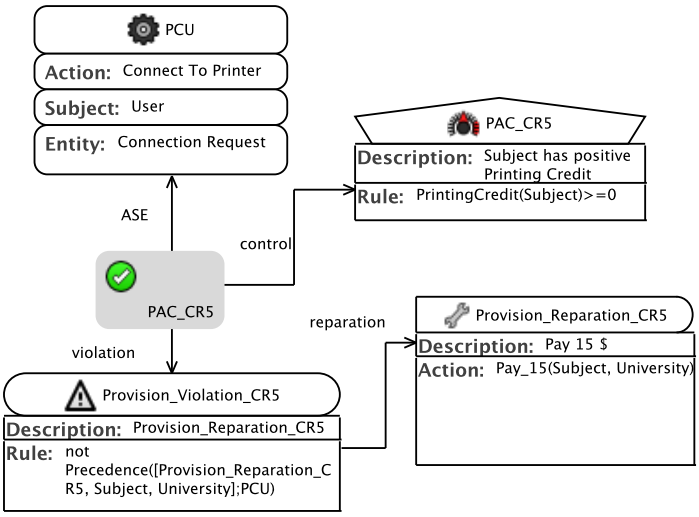


Figure 9.14: CR5 - Policy Model

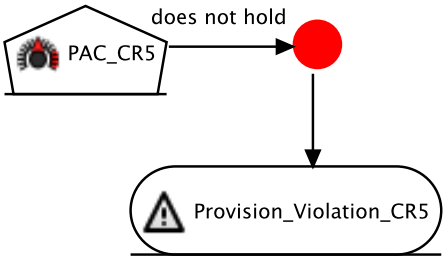


Figure 9.15: CR5 - Violation Model

```
796   all t:Trace | t·precedes[a,b]
797 }
798
799 pred precedesPossibly[a,b: ASE] {
800   some t:Trace | t·precedes[a,b]
801 }
```

The predicate *precedes* models a temporal ordering relation between two ASEs in any trace. It does so by expressing this constraint on the order of occurrence of the two ASEs in a given SBP trace. *PrecedesAlways* tells whether the precedence relation holds in all or in some (at least one) of the execution traces.

In order to visualize what this concretely means for the execution of SBP, we can represent this by having an imaginary third pool, outside of the boundaries of our system, where ASEs are concurrently executed and can be found in the global execution trace. In Figure 9.16 we drew such a pool which represents a hypothetical User X.

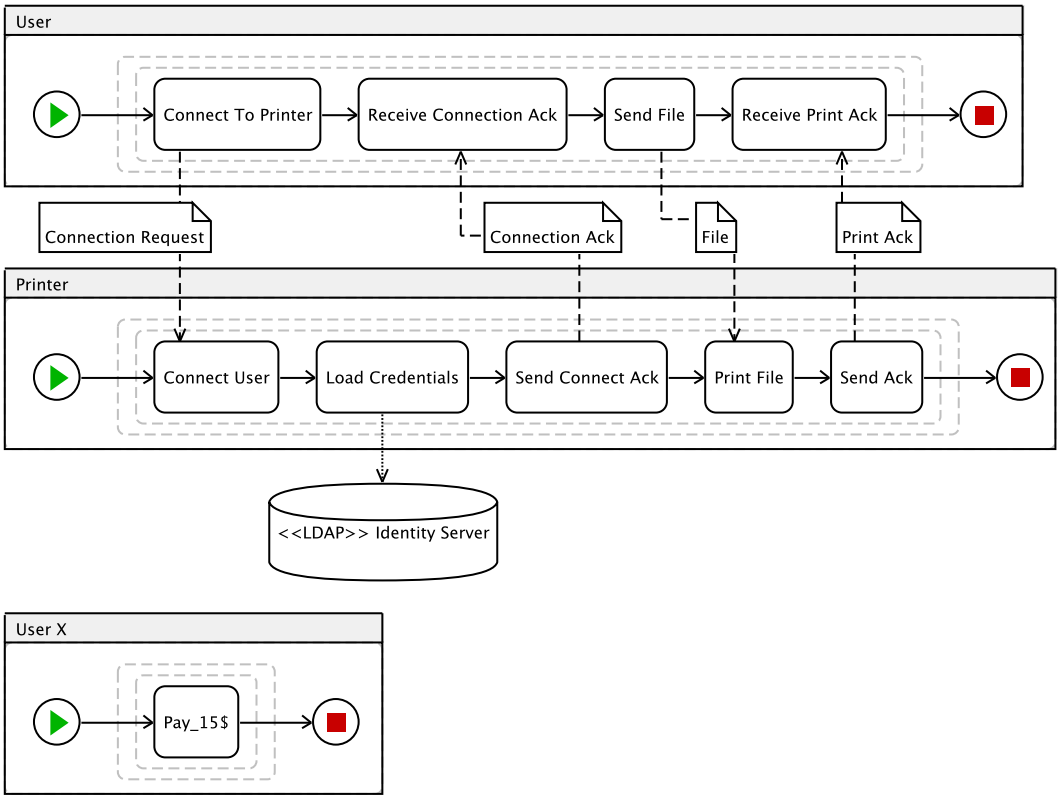


Figure 9.16: The printer Example - Illusrative Enrichment with a Reparation Pool

9.2.3.10 CR6

The last example of CR for the printer example shows in Figures 9.17 and 9.18 how to model a simple reward policy. This policy decides whether or not to allow connections to the printer and admit two kinds of violations. Violation *V3_CR6* shows how to implement the reward and violation *V3_Default_CR6* shows how to implement *Resume* execution decisions by default. Both allowed violations will decide to resume the process in any case.

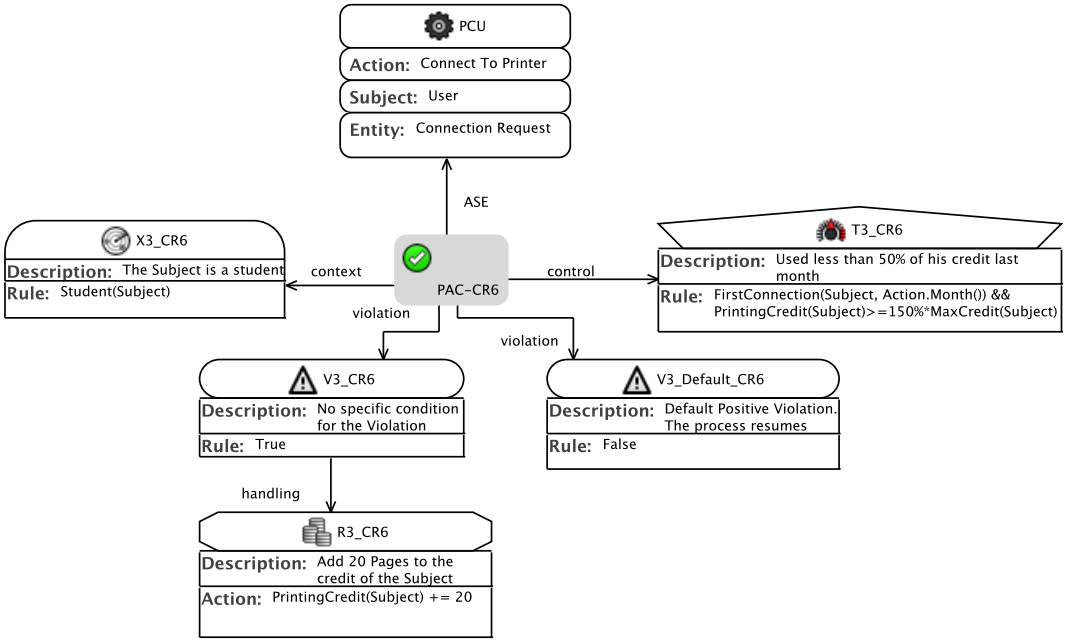


Figure 9.17: CR6 - Policy Model

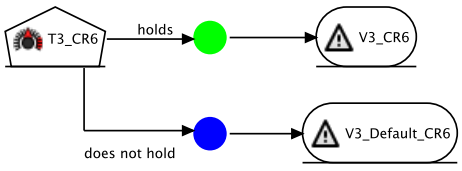


Figure 9.18: CR - Violation Model

9.3 The Expense Refund Case

In this section, we use a different business process model as a support for modeling a new category of CRs. We will target here on the modeling of examples of CRs studied in the literature around compliance, security or organizational control modeling. In this section, we will focus on the expression of controls, rather than additional description of violations and recoveries.

9.3.1 The Process Case

We model CRs for the process model shown in Figure 9.19. This process is composed of three pools, one for an employee requesting a refund, a financial clerk who will process the employee's request, and a manager involved in the validation of the refund request. In this section we use the following abbreviations: RR for Refund Request and RO for Refund Order.

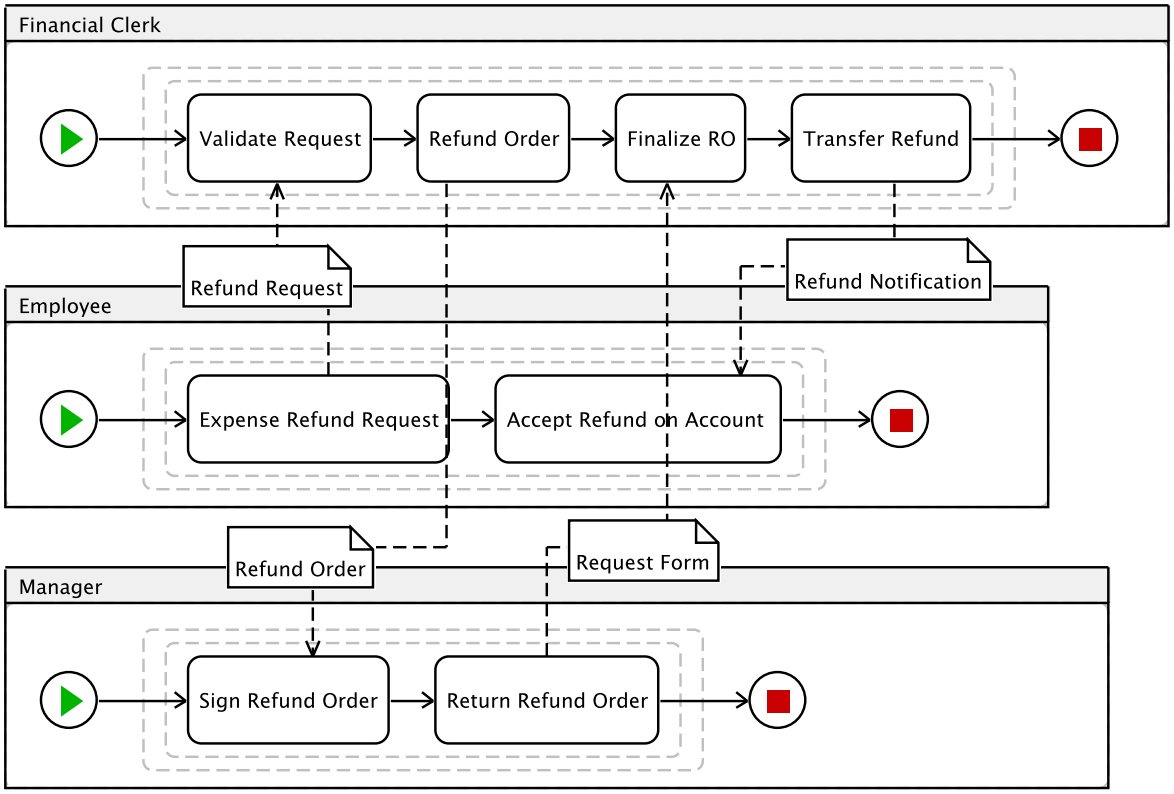


Figure 9.19: Expense Refund Process - in xBPMN

9.3.2 The Compliance Requirements

The CRs for the case are given in Table 9.2. The first CR is a standard static role-based Segregation of Duty (SoD) constraint. The second CR is a location-contextualized and violation-based dynamic SoD. The third and fourth CRs are respectively object- and action-based dynamic

SoD constraints. The last constraint in this set of CRs exemplifies a constraint-based meta-policy constraint.

Table 9.2: Compliance Requirements - Expense Refund Example

CR	Definition
CR7	Static SoD: The Employee requesting the standard refund order cannot be concurrently a Manager.
CR8	Contextual Violation- and Subject-Based SoD: This CR must be complied with throughout our subsidiaries in North America. The constraint of the CR is: If a manager is a staff member and has committed a violation in the past, then he cannot acquire the role of a financial clerk.
CR9	Dynamic Object-based SoD: If an ASE is executed in the past where the ASE's entity is some given object s, and the ASE's subject is another system object e (e.g., some employee) with a role r_x, then this subject cannot have another role r_y.
CR10	Dynamic Action-based SoD: If an ASE is executed in the past where the ASE's action is a and the ASE's subject holds a given role r_x, then this same subject cannot hold another role r_y.
CR11	Static Meta-Policy Constraint (A-/S- or E-based): The policy P states that its ASE is prohibited from execution if this ASE is also governed by another policy P' which is a permission and is in conflict with policy P.
CR12	Dynamic Meta-Policy Constraint (A-/S- or E-based): The policy P states that its ASE is prohibited from execution if this ASE has among its permissions a specific permission P1 in conflict with the current policy and policy P1 was enforced previously in the process.

9.3.3 Engineering the CoReL Models

9.3.3.1 CR7

The static SoD is modelled in Figures 9.20 and 9.21. There are two things to notice about CR7's policy diagram. It has two controls. The first control checks if the role of the ASE's subject holds the role *Manager*. The second control checks if the originator of the *RefundOrder* (the person having requested it) is the same one executing the ASE (its subject). If these two conditions are both true, we know the subject is an employee, since he the *RefundOrder* originated from him, but also that he is a *Manager*.

The violation model of CR7 is basic and is needed to express what should happen when the decision of the policy is not to prohibit the ASE. In case the two controls do not hold, we wish to resume execution. This is why we link this control valuation to a *Green* violation value, itself linked to a *Default_ Positive_ Violation*. The *Green* violation value carries a *resume* policy decision, while the *Default_ Positive_ Violation* has no recovery and a trivially holding rule.

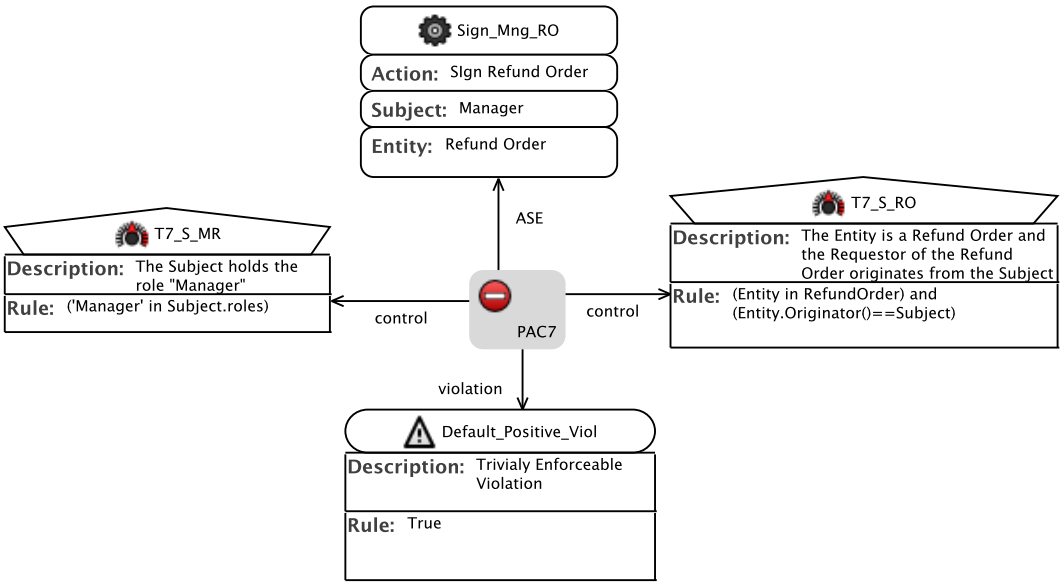


Figure 9.20: CR7 - Policy Model

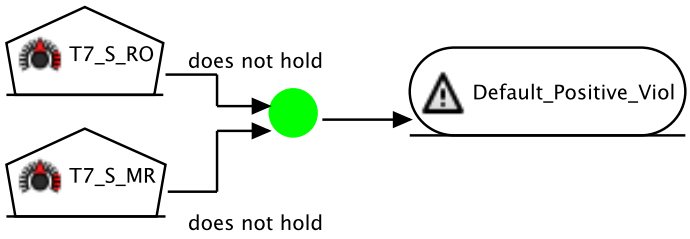


Figure 9.21: CR7 - Violation Model

9.3.3.2 CR8

This CR policy model shows how to use a location based context. The location is a predicate defined for a process instance, and returns the cumulated set of countries where all the tasks of the process take place. The location is a qualifier attached to the action signature, as we show in the following Alloy code.

Code 9.3.1 (Snippet)**.**

```
802 abstract sig Action{
803   location: one Location
804 }
805 abstract sig qualifier{}
806 abstract sig Location extends qualifier{}
807 one sig usa extends Location{}
808 one sig canada extends Location{}
```

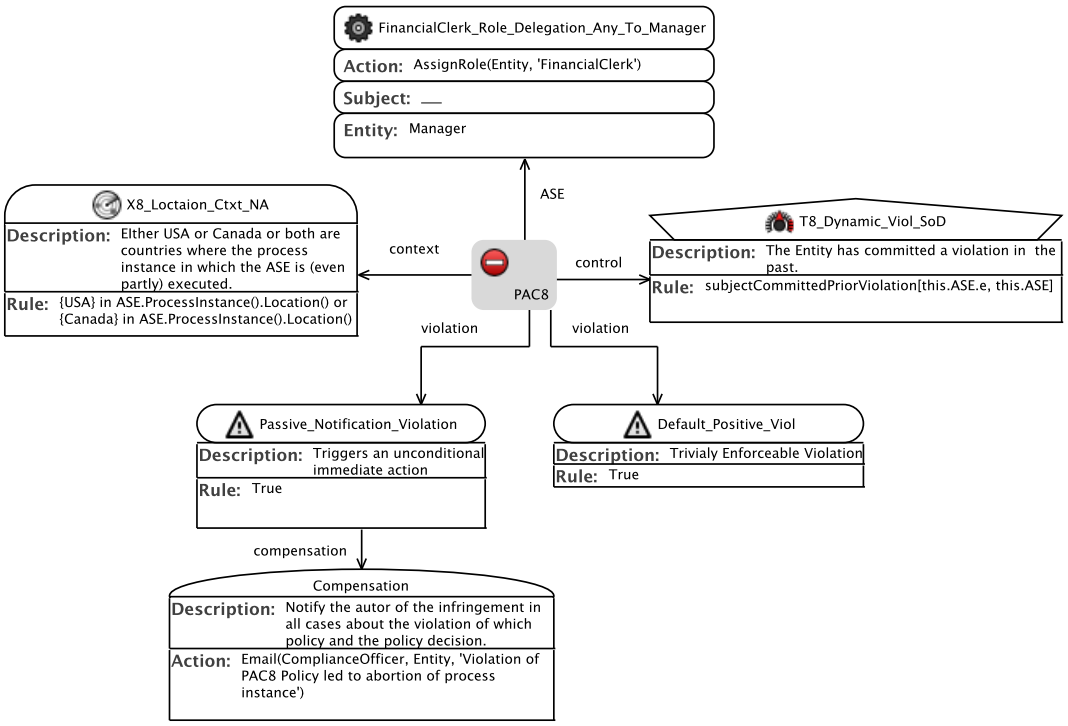


Figure 9.22: CR8 - Policy Model

The key to modeling CR8 is to model the right ASE. Here we are concerned about ASEs occurring outside the boundaries of the Expense Refund process, which is a role assignment ASE, assigning the *FinancialClerk* role to a person already carrying the role *Manager*. Such ASEs may happen at any time and can be modeled as ASE events logged in the CoReL ComplianceTrace.

Once the right ASE is modelled, we model a context for North America. We Assume every Action in the system has a qualifier which carries its location (i.e., geographical location where the action is performed). This predicate returns true if at least one of the actions in the system

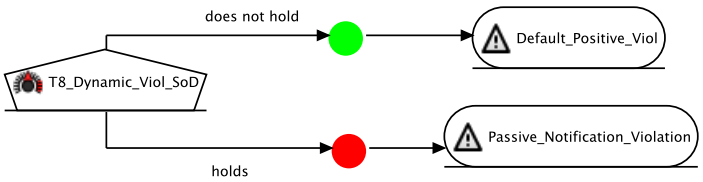


Figure 9.23: CR8 - Violation Model

model takes place in North America.

Another useful predicate used in the modeling of this policy is given by *subjectCommitted-PriorViolation(Subject, ASE)*. This Alloy predicate holds when the subject committed a prior violation, i.e., when the subject belongs to an ASE governed by a policy, which was in a violated state previously in the trace. The pattern we are looking for in the trace is represented in Figure 9.24.

Code 9.3.2 (Snippet)**.**

```
809 pred ComplianceTrace::subjectCommittedPriorViolation[sub:Subject, se:ASE] {
810   some i : this·states·inds | some p:Policy |
811     this·states[i]·policyState[p] in violated and (sub = p·governs·s) and precedesAlways[p·governs,se]
812 }
813
814 pred subjectCommittedPriorViolation[sub:Subject, se:ASE] {
815   all t:ComplianceTrace | some i : t·states·inds |
816     (all an:AseNode | t·states[i]·nodeState[an] in active) ⇒ t·subjectCommittedPriorViolation[sub,se]
817 }
```

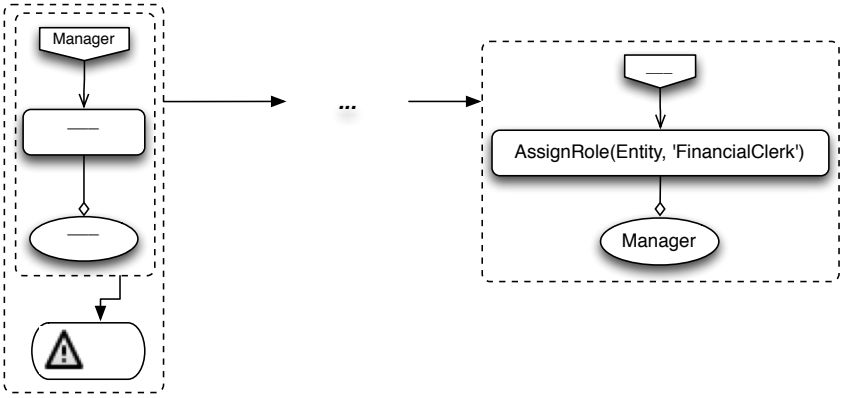


Figure 9.24: CR8 - Illustration of the *subjectCommittedPriorViolation(Subject, ASE)* predicate

9.3.3.3 CR9 & CR10

For these CRs we only show the Policy model. The interesting part are the Alloy predicates required to model the control part of CR9 and CR10.

Figure 9.25: CR11 - Policy Model

Before we can introduce the predicates key to modeling the controls in Figure 9.25, we must first show how how we can enrich the ASE model with information about roles. We explained this in Sections 5.4.1.2 and 8.2.1. All that is needed is a new relation between the signature subject and the role signature.

Code 9.3.3 (Snippet)**.**

```
818 abstract sig Subject extends Object{
819   roles: set Role
820 }
821 abstract sig Role extends qualifier{}
822 one sig r1 extends Role{}
823 one sig r2 extends Role{}
824
825 pred Trace::precedesASEWithRole[a,b: ASE, r:Role] {
826   all j: Int| all n2: AseNode| n2 in this·activeSets[j] && n2·se =b ⇒
827   some i: Int| some n1: AseNode |
828     i< =j && n1 in this·activeSets[i] && n1·se = a && (r in (n1·se)·s·roles)
829 }
830
831 pred precedesASEWithRoleAlways[a,b: ASE, r:Role] {
832   all t:Trace | t·precedesASEWithRole[a,b,r]
833 }
834
835 pred Trace::precedesASEWithRoleAndAction[ase1,ase2: ASE, r:Role, ay:Action] {
836   all j: Int| all n2: AseNode| n2 in this·activeSets[j] && n2·se =ase2 ⇒
837   some i: Int| some n1: AseNode |
838     i< =j && n1 in this·activeSets[i] && n1·se = ase1 && (r in n1·se·s·roles) && (ay = (n1·se)·a)
```

```
}

pred precedesASEWithRoleAndActionAlways[a,b: ASE, r:Role, ay:Action] {
  all t:Trace | t.precedesASEWithRoleAndAction[a,b,r,ay]
}
```

Four different predicates are shown in code snippet 9.3.3. The first predicate *precedesASEWithRole(ASE1, ASE2, Role)* holds when $ASE2$ is preceded in the trace with the execution of an $ASE1$ whose subject has a role *Role*. The third predicate *precedesASEWithRoleAndAction(ASE1, ASE2, Role, Action)* holds whenever $ASE2$ is preceded by the execution of an $ASE1$ where the action is equal to *Action* and the subject carries a role *Role*. Predicate number 2 resp. 4 is the generalization of predicate number 1 resp. 3 to all the possible traces.

9.3.3.4 CR11 & CR12

CR11 resp. CR12 are referred to as static resp. dynamic conflict-based meta-policies. These requirements basically allow to reason on relations amongst policies. For instance, in some regulations, a given policy may be designed as being mutually exclusive with a set of other policies, i.e., in conflict with them.

By conflict, we do not mean that while reasoning on both policies, we might have situations where we arrive to a conflict in terms of decisions by the policies (e.g., policy P1 decided to permit an ASE while policy P2 decides the opposite). We mean conflicts modelled by the user, who does not want that both policies apply to a set of ASEs during the same execution. For example, we may have a generous extra holiday policy applying to employees of a company in conflict with another policy applying generous financial compensation to employees having shown great commitment in terms of extra hours spent at work during hot project phases. The reason is that the company may not want its employees to have to choose between money and free time.

In order to be able to model the controls in Figure 9.26, we must first extend the definition of the policy signature. We add an non-reflexive relation *conflicts* from a policy to a set of other policies.

Code 9.3.4 (Snippet)**.**

```
abstract sig Policy{
  modality: one Deon,
  governs: ASE,
  context: Constraint,
  control: seq Constraint,
  violation: seq Violation,
  abort: Bool,
  conflicts : set Policy
}{
not (this in this.conflicts)
}
```

Then we can model the desired predicates used in Figure 9.26 for policies CR11 (named *CR_MetaPolicyConflict*) and CR12 (named *CR_MetaPolicyConflict_Past*) by reusing the *precedes* and *follows* predicates. The predicate *hasConflictingPermission(ASE, Policy)* statically checks the existence of a permission policy P2 governing the same ASE as a given policy P1.

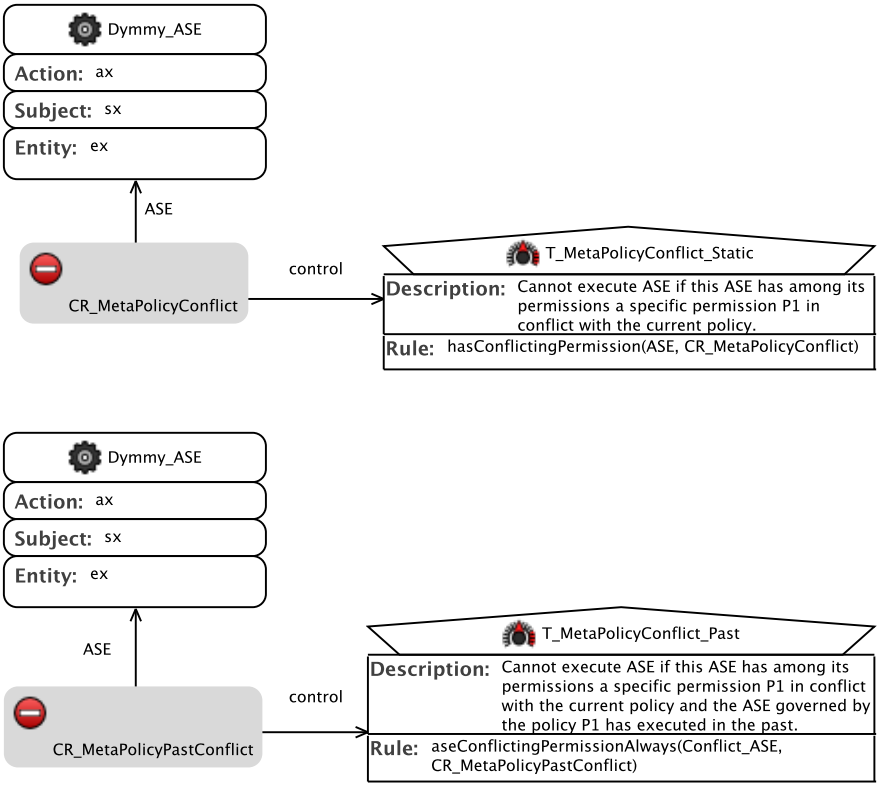


Figure 9.26: CR11 & CR12 - Policy Models

This information is accessible without executing the system, ans is retrieved using the *conflicts* relation previously modeled for policies.
The more complex variant is predicate *ComplianceTrace::aseConflictingPermission(ASE, Policy)*, which implements the dynamic checking in the prior segment of the ComplianceTrace for an activated permission policy. If this policy P1 is in conflict with a policy P2 and applies to the same ASE, then the predicate holds.

Code 9.3.5 (Snippet)**.**

```
855 pred aseConflictingPermission[ax:ASE, px:Policy]{
856   some p:Policy | (ax in p.governs) && (p.modality = Permission) && (p in px.conflicts)
857 }
858
859 pred ComplianceTrace::aseConflictingPermission[ax:ASE, px:Policy]{
860   some i : this.states.inds | some p:Policy | this.states[i].policyState[p] in active &&
861   (ax in p.governs) && (p.modality = Permission) && (p in px.conflicts)
862 }
863
864 pred aseConflictingPermissionAlways[ax:ASE, px:Policy]{
865   all t:ComplianceTrace | t.aseConflictingPermission[ax,px]
866 }
867
868 pred aseConflictingPermissionPossibly[ax:ASE, px:Policy]{
869   some t:ComplianceTrace | t.aseConflictingPermission[ax,px]
870 }
```

9.4 Modeling an Article of the HIPAA Regulation

As a last part of this chapter, we aim at getting an approximate idea of the remaining challenges of modeling real-world regulations using CoReL. We do this by applying CoReL to a real-world regulation. The following text is a small excerpt from a regulation from the healthcare domain on privacy rules called HIPAA. The HIPAA (US health Insurance Portability and Accountability Act of 1996 ("HIPAA") - Privacy and Security Rule) is a regulation that protects the privacy of an individual's health information and govern the way certain health care providers and benefits plans collect, maintain, use and disclose protected health information in the United States[1].

The difference between this regulation and the previous two examples we used, is that it is more abstract and does not refer to any concrete process steps or known enterprise model elements. This regulation deals with specifying which information on individuals that covered entities possess (i.e., health insurance entities) might be disclosed and under which conditions.

In this section, we will evaluate the degree to which the modeling constructs of CoReL allow to cover elements of a real-life regulation. For this, we will not include predicates in the modeling, and content ourselves with informal textual modeling to lead the discussion.

Example 9.4.1 (HIPAA Regulation[2])**.**

§ 164.512 Uses and disclosures for which an authorization or opportunity to agree or object is not required.
A covered entity may use or disclose protected health information without the written authorization of the individual, as described in §164.508, or the opportunity for the individual to agree or object as described in §164.510, in the situations covered by this section, subject to the applicable requirements of this section. When the covered entity is required by this section to inform the individual of, or when the individual may agree to, a use or disclosure permitted by this section, the covered entity's information and the individual's agreement may be given orally.

(f)Standard: *Disclosures for law enforcement purposes. A covered entity may disclose protected health information for a law enforcement purpose to a law enforcement official if the conditions in paragraphs (f)(1) through (f)(6) of this section are met, as applicable.*

(f)(1) *Permitted disclosures: Pursuant to process and as otherwise required by law. A covered entity may disclose protected health information:*

(f)(1)(i) *: As required by law including laws that require the reporting of certain types of wounds or other physical injuries, except for laws subject to paragraph (b)(1)(ii) or (c)(1)(i) of this section; or*

(f)(1)(ii) *: In compliance with and as limited by the relevant requirements of:*

(A) *A court order or court-ordered warrant, or a subpoena or summons issued by a judicial officer;*

(B) *A grand jury subpoena; or*

[1] http://hipaa.stanford.edu/

[2] See http://www.hhs.gov/ocr/privacy/, http://hipaa.stanford.edu/, and http://privacyruleandresearch.nih.gov/.

> **(B)** *An administrative request, including an administrative subpoena or summons, a civil or an authorized investigative demand, or similar process authorized under law, provided that:*
>
> **(1)** *The information sought is relevant and material to a legitimate law enforcement inquiry;*
>
> **(2)** *The request is specific and limited in scope to the extent reasonably practicable in light of the purpose for which the information is sought; and*
>
> **(3)** *De-identified information could not reasonably be used.*

When attempting to model such a regulation one may be tempted to start looking at each of the nested statements and try to represent them using logic or our formal model. Using CoReL we know we must start by looking at what kinds of ASEs are concerned by each statement. Then we can look at how statements in the regulation applying to this ASE may be grouped in a way that makes sense. We need to do this even though we have no business process to check compliance of for the moment. This is one of the advantages of using ASEs.

If we start with the CR **§164.512**, we understand that the context of the regulation are all covered entities in the United States. Moreover, we deal with several ASEs here. The first ASE is the covered entity disclosing information about the individual to a third party: *(Disclose, Covered Entity, Information about the Individual)*. The second ASE is the notification of the individual of this disclosure: *(Notify of Disclosure, Covered Entity, Individual)*. The third ASE is whether or not the individual agrees or objects to this disclosure: *(Agree/Object, Individual, Disclosure)*. We may add these two ASEs to the list: *(DiscloseOrally, Covered Entity, Individual Information)* and *(AgreeOrally/ObjectOrally, individual, Disclosure)*.

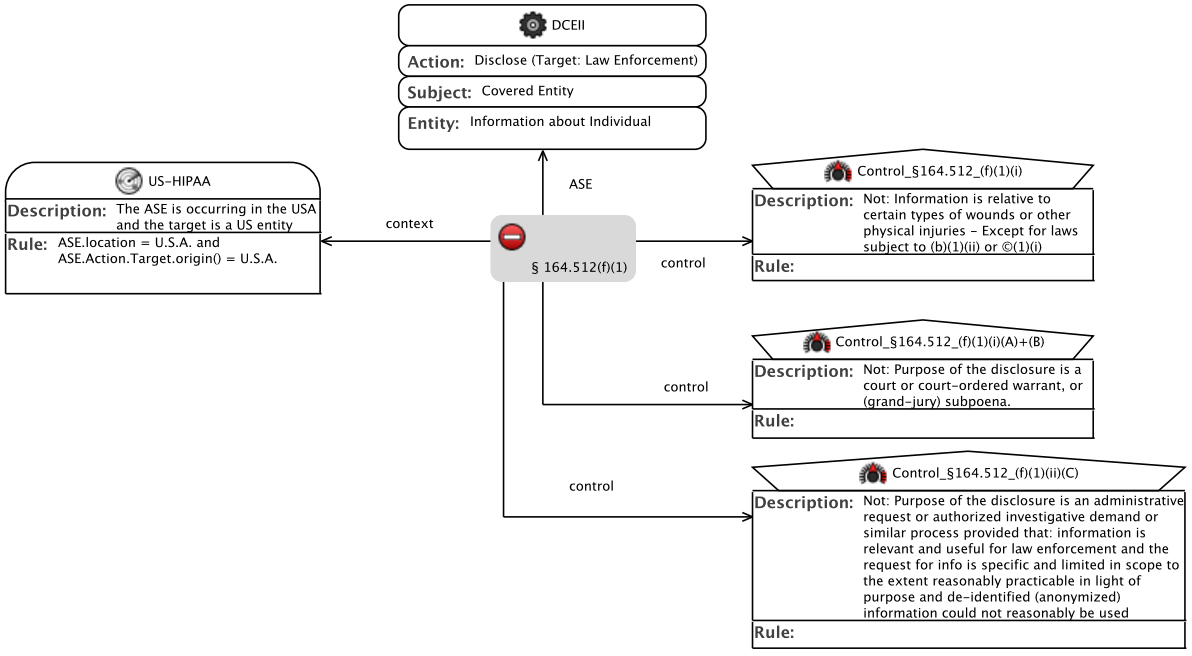


Figure 9.27: Informal Proposal to Model Article §164.512(f) of HIPAA

When attempting to collect and organize the elements of the regulatory text that might help us formally model the compliance requirements included in the regulation, we encounter

many problems. First, we notice that some metadata about the Action is needed, particularly about the Target of the Action, and its Purpose. Also, more information about the execution environment in which information about individuals is requested and disclosed is required. For instance, questions have to be answered such as:

- What types of wounds and physical injuries are meant?
- How to judge if the information is relevant to law enforcement? What information is critical to the health and security of the individual?
- What types of information is there and how is it organized? How does the covered entity possess about the individual? Where and in which form is this information stored?

Moreover, the regulation refers to exceptions to the rules (CoReL controls) which are specified in separate articles of the law. CoReL does not support exceptions to a rule. Such an exception must be processed by a knowledgeable legal expert by hand. Additionally, several elements of the regulation must be interpreted and concretized for a particular execution environment (e.g., a company, a hospital, a dentist office, a health insurer, etc.). All of these make formalizing the regulation at this level of abstraction for the sake of automated checking a hard task.

In summary, these challenging aspects are some of the research topics that need to be studied in future work on CoReL.

Part IV

Finale

Chapter 10

Conclusion

A master and his disciple were travelling through the desert. One evening they came to an oasis were they bedded down for the night. When they awoke in the morning, their camels were gone. Since thethering the camels every night was the disciple's responsibility, the master asked him whether he had secured them for the night. The disciple replied: "No master, you always teach me that we should trust in Allah. I was trusting that Allah would take care of the camels for us". To this his master replied: "Yes, trust in Allah, but you must also tether the camels".

Ancient Arab wisdom story. From "When working on yourself doesn't work". By Ariel and Shya Kane. 2009.

This chapter starts with Section 10.1, where we first summarize the research work in Section 10.1.1, followed by an elaboration on the research results in Section 10.1.2. Then, we present the contributions of the dissertation in Section 10.2. Section 10.3 groups three separate items, as we describe lessons learned in Section 10.3.1, limitations of the work in Section 10.3.2 and give a comparison to existing research in Section 10.3.3. Potential future work and interesting challenges are discussed in 10.4 and Section 10.5 closes this thesis.

10.1 Research Summary

10.1.1 Synopsis

In the context of business process-centered enterprise models, ensuring compliance to various regulations is a very critical property. Much of the research on RCM focuses on developing rule modeling languages and verification approaches for implementing RCM. Our analysis of regulations and the needs in business process-centered enterprise modeling has shown that one of the untackled needs of RCM is the adequacy of the approach for use by business users.

Also, the ability to simplify the CR modeling process and reduce the required effort to create CR models are additional criteria which might help increase the acceptance of RCM among business users. This is even more important if we consider the environment in which enterprises evolve. We have also argued that there is a semantic divide between the compliance requirements domain and the constraint/rule domain. Also, we have expressed the need for support of various enterprise artifacts concerned by RCM.

Both regulations as well as enterprise models are subject to regular change. We have argued the benefits of a visual DSL for representing CRs which allows to create models by building on

reusable components. The CoReL language allows us to reason on multiple violations and comes equipped with formal semantics for both verification and enforcement. We have shown how the latter two functionalities can be integrated within business process-centered enterprise modeling. Finally, the contributions of the thesis have been showcased in case studies and using developed tool support, thereby showing the feasability of the approach and its applicability to RCM for BPM.

10.1.2 Results

This section recapitulates the research questions identified in Chapter1, and arguments why and how the work performed in this thesis answers these research questions.

RQ 1: How to make RCM Modeling more amenable to Business Users? The main challenge that business users have to deal with is to possess appropriate means to express their understanding of CRs. We tackle this challenge by proposing to abstract from the rules in CoReL policies. We assume that business users are able to select appropriate rules based on their description.

RQ 2: How to exploit the power of logical formalisms for compliance management? Our proposition to answer RQ2 is to embed rules into carrier concepts (contexts, controls, violations). Thereby, we can theoretically make use of any appropriate formalism to check a rule. Our answer to RQ2 is limited, as we only showed this for Alloy, but not for other formalisms, which would have given CoReL support for multiple formalisms.

RQ 3: How to cover several types of enterprise business artifacts? We tackled RQ3 by defining ASEs as the core paradigm to model actions happening in the system. Together with business modeling extensions using Qualifiers and ASEs, we can represent rich business model properties of targets of policies. We showed this in examples which required to model policies on roles or on sending messages in Chapter 9.

RQ 4: How to decouple compliance modeling from BPM notations? RQ4 was also tackled through ASEs together with SBPs to yield a formalism for system modeling. The ASE triple is independent of any specific BP modeling language and our work hypothesis was that it is possible to extract ASE triples from a BP modeling language. This is how CoReL is decoupled from BP notations. We illustrated this point for a prototypical BP modeling language strongly inspired from BPMN in Section 8.1 where we allowed for including additional non-workflow artifacts into modelled policy rules.

RQ 5: How to realize *'verification'* using compliance models? The formal semantics of CoReL models is given in Chapter 6. These operational semantics are defined declaratively in Alloy over a logic theoretical model based on system (SBP & ASE) and policy execution traces. The interesting property of these semantics is that they allow to reason on ASEs as well as on policy states. This allows us, for example, to reason on violations.

RQ 6: How to realize *'enforcement'* using compliance models? This RQ is strongly tied to RQ5, since the formal semantics of CoReL defined in Chapter 6 also defines a policy execution machine (operational semantics). This means the semantics describes enforcement by simulation, as it acts as a formal specification of an interpretation machine that merges the execution of CoReL policy models together with system models.

RQ 7: How far can the Model-Driven Engineering discipline support our endeavour? We used metamodeling and model transformations to equip the CoReL DSL with an abstract syntax and formal semantics. We used the ECORE [Riv10] framework on the Eclipse platform with its EMF/GMF tooling to define the concrete syntax and the diagram editors we contribute within this thesis.

10.2 Contributions

In [HMPR04], the objective of design science research is said to be: *"to develop technology-based solutions to important and relevant business problems"*. [HMPR04] also states that *"design science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation"*. Aligned with the previous requirements from design research theory, we have contributed three design artifacts: a modeling language, a verification approach, and a supporting tool.

In this thesis, we have tackled the study of regulatory compliance management RCM from the perspective of business users. Business users are legal or business model/process experts. We have analyzed a comprehensive set of existing approaches to RCM and have elicited a conceptual framework for RCM in BPM which allows to have an overview of research to structure and compare research in this field. This framework contains a list of 33 different criteria organized in 8 categories.

We have designed and formalized a core formal model for business process-centered enterprise modeling in Chapter 4, referred to as system modeling, also following an MDE approach to define a prototypical language we call xBPMN (see Chapter 8). We have contributed the model-driven CoReL approach. This approach follows the MDE paradigm in engineering the CoReL modeling language (see Chapters 5 and 6) and designing its semantics and tooling (visual editors) as reported in Chapter 7. Based on that, we have shown how to integrate the CoReL language both syntactically and semantically with the system modeling language in Chapter 6. Finally, a tool suite built using state of the art model driven engineering technologies has been implemented[1] as described in Chapter 7.1.

10.3 Discussion

10.3.1 Challenges Faced & Lessons Learned

In this section, we discuss some of the most important lessons learned from our research on the CoReL approach for model-driven regulatory compliance modeling and verification. While the objective of the research was well-defined and confined to research on the feasability and user-friendliness of a formal MDE-driven and policy based solution, the problem we tackle led us

[1]The implemented tool is the result of team effort, in the scope of the FNR MaRCo project, and not the effort of the thesis author only.

to be confronted with several related problems pertaining to DSL engineering and usage of the developed framework.

Designing a DSL or designing a semantic bridge?
A critical mind might say about this work that we designed a DSL for a domain we are not experts in. It is true because we propose a DSL for ***legal and business domain experts to use***. But CoReL is not the only DSL that legal/business experts are going to use, as CoReL has a specific purpose of modeling compliance decision-making.

CoReL's use is not solely for knowledge representation (e.g., documentation or refinement), or knowledge retrieval (e.g., using Ontologies as we published here [EKP09]). CoReL maps the modeled decision-making into the formal methods/logics domain where the actual automated reasoning on regulations is conducted. Thus, it is a semantic bridge DSL.

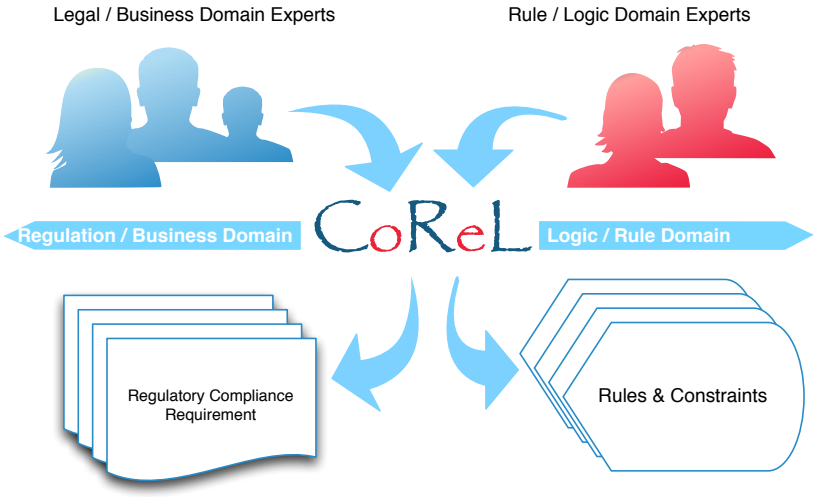


Figure 10.1: CoReL: A DSL or a Semantic Bridge?

Keeping it Small
One objective was to keep the number of concepts in CoReL as small as possible. The more concepts there the higher the conceptual complexity and the risk for redundancy and ambiguity. Also, the time required to learn the language increases. We limited ourselves to the bare minimum of useful and semantically distinct concepts. For example, we have contexts and controls, although both are constraints in the target semantic domain (Alloy). The same reason led us to make rewards and penalties into two different concrete syntax occurrences of the same metamodel and semantic domain concept: handling.

Designing a Suitable Visual Concrete Syntax
Second, designing a visual language involves a whole range of skills around cognitive, ergonomic and technical aspects of language design that was not the core set of skills we possess. Moody has studied extensively this complex problem and approaches to deal with it in a set of works named 'physics of notations' [MvH09, Moo09, Moo03].

However, we learned that usage of discriminating shapes and explicit icons helps in quickly distinguishing symbols from one another. We also learned that the notation commonly used in

software engineering modeling languages of compartments inside shapes to visualize properties (e.g., in UML or ECORE) is very useful for embedding much information into a visual shape. We also experimented with using colours for the shapes as well as icons on the associations between concepts. We found that this can lead to an overload of the diagram, due to the number of various concepts in the CoReL language. In the final versions of our tool, we reverted back to simple black and white colors and straight lines, as we found it to be easier on the eye.

Dealing with Great Numbers of Policies and Policy Diagrams
First of all, modeling CRs as business policies leaves room for some interpretation, even when the CRs are very concretely described. Depending on the preferred modeling style of the user, there could be more than one way of representing one CR. Next to the fact that visual policy and violation diagrams can get very big and complicated, there is the phenomenon that for a regulation with n compliance requirements we will have at least $2 * n + 1$ models (+1 because of at least one regulation model).

This might be quite overwhelming to the modeler and a sense of being 'lost in a sea of models' can result from this. We need to tackle this issue by studying further means of semantically organizing policies into a repository. The possibilities are many, e.g., by type of ASE the policy governs, by process that should comply with it, by person accountable for it, by number of activations or violations, or by type of constraint it expresses, etc. There is a rich set of possible approaches to this problem, ultimately leading to extending the enterprise model with a business policy architecture in the enterprise.

Using Rules in CoReL Shapes
Our experience with CoReL has shown us that it is somewhat hard to foresee the reaction of business users to using strings carrying rules in control/context/violation shapes. We have conducted a limited number of informal interviews with legal experts and their feedback to the language concepts was positive, whereas they complained that the rules embedded in CoReL symbols look too complex. They felt they could express the same more easily in text, thereby letting go of the promise of automated checking.

We speculated that the legal experts might feel uncomfortable holding responsibility for creating policy models in which they have to rely on textual description of the rules to know which rule to choose. This is the reason why we advise in the introduction of CoReL to let legal experts choose the rules they want to embed in policy shapes together with logic or computer science experts. However, languages such as SBVR [Gro13], which is a textual business rule language in structured natural language designed specifically for business users might help. A language like SBVR could help replace the first order logic rules we use in this thesis with SBVR statements which can be later translated into a logic (like Alloy).

10.3.2 Limitations of the Work

10.3.2.1 Limitations related to the Research Questions

Policy management does not only mean checking policies.
The other challenge related to RQ1 (cf. Section 10.1.2) is the daunting task of creating, testing and managing policies. This is not a trivial task because of several factors: (i) size of regulations and number of various policies, (ii) effort in creating models from scratch, (iii) organizing policies,

keeping an overview and structuring them to relate them to the regulatory text in order to ensure traceability. We partly tackle this issue by designing CoReL to be a modular language. This allows to create policy models by reusing components (such as controls, violations). We proposed regulation diagrams to tackle the latter three success factors. However, we did not validate this aspect of the work, by for instance, modeling a regulation great in size and introducing changes.

Using multiple rule formalisms.
There is also a limitation related to RQ2 (cf. Section 10.1.2). In theory, nothing limits us to using Alloy for checking a rule, as long as there is an engine capable of evaluating this rule to either True or False. In order to check rules in various languages, the system model must be transformed into the required formalism for the rule to be checked, and provided there is an interface (API) between Alloy and the rule engine in question, so that evaluating a rule in a foreign formalism to Alloy could be implemented. But this would require us to have access to the Alloy implementation itself and to extend it.

On rules impact on the usability of CoReL.
Another direct limitation of the work is also related to the rules in CoReL. Although this is not a central limitation as it was not one of the objectives of the research, it still has a considerable influence on the usability of the approach. We applied CoReL to our examples while abstracting from the complexity of the task of selecting and understanding rules embedded in CoReL elements such as context and control. We think this limits the usability of CoReL at this stage.

Our intended direction of research for solving this problem is developing a visual pattern-based language of rules expressing constraints on ASEs and their attached qualifiers. These rules can be interpreted by each policy differently depending on the ASE it governs and which is active at the current state of process execution. The work by Awad in [Awa10] is the one which is also going the same direction, although the approach used is different. In this work, BPMN artifacts used by (as input or output) process tasks are annotated with states. These states must be considered in the formulation of the rules to be checked and allow to express rules. In [Ama12], the main contribution of the thesis is a pattern-based visual rule language of formulas translated into LTL. A similar idea is pursued in [XLW08]. This research is one of the group of works which build on the property specification patterns [YMH+06, MGJ98, GL06].

10.3.2.2 Limitations related to the Research

CoReL's support of interleaving in branches' execution.

One limitation is that we did not consider arbitrary interleaving in the execution order of tasks belonging to separate branches in the system (enriched process) model. For example, two tasks defined after an AndSplit are always executed concurrently by our Alloy engine. This is obviously not always the case in reality. In fact, research has shown that this interleaving in execution order in branches is the main reason for the explosion in complexity of the checking problem ([CTGKvdT12, CTEKG+13], where the problem is shown to be hard). Therefore, it would be interesting for us, when investigating the complexity of the compliance checking in BPs problem, to evaluate the performance of the Alloy tool with this regard.

CoReL's validation.
It is certainly a limitation of the contributions around CoReL that we didn't conduct field studies answering questions about the complexity and usability of the language. Another objective of a comprehensive and robust validation would have been evaluating the practicability of using CoReL for real-world compliance modeling and checking, in terms of performance. Also of value would be defining the parameters of the case (size of the process, complexity of the workflow, number of artiacts used, complexity of the rules in contexts, controls and violations) to find the limits of using Alloy as a lightweight formal method. In other words, we want to answer the question: starting from which threshold does using the Alloy analyzer become impractical since it involves a time duration for the checking unbearable in practice?

Using Alloy for formalizing CoReL.
We have made extensive use of the Alloy analyzer, both for validating the semantics as these were still under development, but also for simulating system and policy executions. We have come to conclude that the interface of the tool and the concrete syntax it uses to represent atoms are not very helpful when developing a complex semantics. What we would have needed were counter-examples and generated instances to be directly visualized as process flows and policies, using a similar concrete syntax to the one we have used to introduce SBPs.

This is why the group where this thesis was conducted started a new thread of research which seeks to declaratively assign concrete syntaxes to generated Alloy instances [GK14]. This way, the work of the software language engineer developing semantics for her language will be easier, as she will be able to interact with Alloy instances using a syntax related to her problem and which she understands more readily.

10.3.3 Comparison with other Approaches

In Chapter 3, we surveyed existing approaches to RCM. Here we elaborate on what distinguishes our work from these approaches. Each of these approaches has many strengths. CoReL however, focuses on the semantic divide between the domain language use in the regulatory and business domain on the one side, and the logics/rule language on the other (cf. Figure10.2).

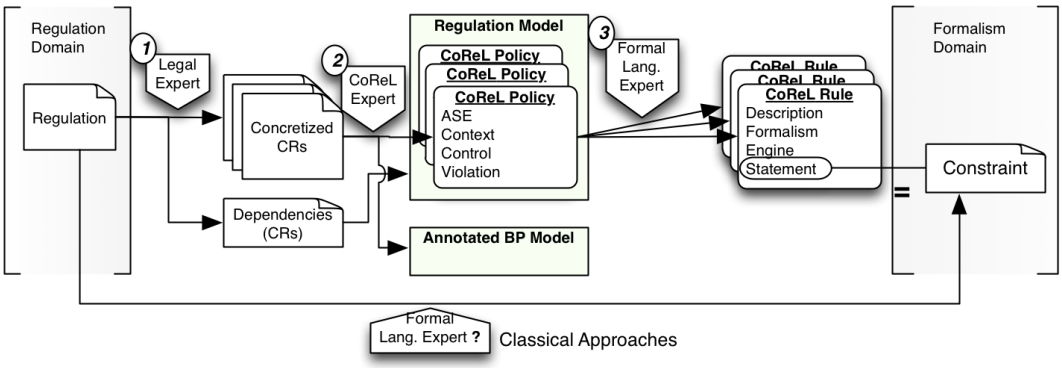


Figure 10.2: Resolving the RCM Semantic Divide

Compliance elements as first-class citizens of the language.

None of the language-based approaches we surveyed deals with elements of compliance as separate first class citizens. By first class citizens of a language, we means concepts that the language natively allows to represent and reason about. An example of this is the ability of reasoning on violations, both statically and dynamically, as shown in the case study chapter.

What does business user-friendliness really mean?
Most of the approaches which also recognize the problem of adequacy of the solution to business users propose a visual and/or pattern-based rule language. Although this is certainly a helpful thing to simplify and speed up expressing rich business rules, it still is at one semantic level too deep. Doing so misses the point of enabling business users to model and document the CRs using domain concepts which appeal to them and directly relate to their conceptualization of RCM. However, we see such approaches as a promising extension of CoReL as using visual and pattern-based business rules for expressing constraints embedded into CRs would probably ease the modeling by business users. We mention this aspect in the limitations of the thesis.

Modularity.
Another difference between CoReL and existing RCM research is the strong modularity that is enabled because of the decomposition of the language in separate reusable constructs. Modularity is a well-established strategy to reduce complexity. This helps make building visual policy models an easier task. Of course, this would be more strongly supported by a repository of these constructs, i.e., a repository of rules and recoveries for example. This is for example not the case in the other approaches, to the best of our knowledge. Even the pattern-based rule languages do not specify how far rules can be composed with each other and what semantics would be defined for this.

Using CoReL with other notations for process modeling.
Because it uses ASE as the most fundamental element to represent system actions, CoReL can abstract from the concrete enterprise modeling or the business process modeling language it is used with. This is achieved because CoReL only models the concepts it needs for its inference, Actions, Subjects and Entities, and no concepts or property from external semantic domains, such as role, resource, message or database. Concretely, that means that the same CoReL model can be used identically with two distinct BP modeling languages, provided that a semantics in terms of ASEs and SBPs is given to the languages. This is why we claim that the modeled policies in CoReL may be used in conjunction with various process modeling languages (module the limitation to SBP-like workflows).

For many of the surveyed approaches, either a specific language is used, or it is unclear from the publications whether they depend on a BP modeling language or not, while for some others, this is not a requirement. The work in [Awa10, LSG07] represent two other ways of ensuring orthogonality with regard to the process language. In [Awa10], semantics to a subset of BPMN is given in terms of P/T Petri Nets, and a model checker for Petri Nets is used to check temporal formula in CTL generated from the visual rule notation Awad proposes. In [LSG07], the approach is very different, as the key to rule interpretation are mostly manual annotations of process tasks with propositions. These annotations are used by the checking algorithm which traverses the process model to verify compliance. Theoretically, it is our understanding that the algorithm used is not dependent on a single process notation.

10.4 Perspectives for Future Research

We organize the discussion along five dimensions: (i) policy, (ii) decision, (iii) rules & verification, (iv) business process and (v) language engineering.

- **Policy Dimension:**

  Obligations are an immediate extension to our work, and we see obligation policies as the most interesting exension. Obligations are a core element of compliance requirements which we do not tackle in the thesis. We see the stream of work realized by Governatori [GM05] (resp. Governatori et al.) as the most promising starting point for this research.

  Policy Algebra Next to this, we initiated work on modeling complex policies, by defining algebraic operators and their semantics, as well as temporal policies. Also, we initiated research on the semantics of these policy composition operators, thereby defining an algebra on policies.

  Decision Flows We also initiated research on the semantics of so-called decision flows, which are workflows of policies. We estimate that this further development of the modeling capacities of CoReL would make the language more useful.

- **Decision Dimension:**

  Policy Activating Violations: An interesting feature worth investigating is enabling violations to request the enforcement of policies. This makes the CoReL language more expressive and allows to cover more complex decision-making patterns.

  Context/Control Algebra: Being able to model complex contexts and controls by using specifically defined algebraic operators will allow policy modelers to represent more complex activation and constraints of policies. A possible start is the work in [ECCB11] which presents an interesting logic for defining expressive contexts by using event-based semantics also supporting activation and deactivation of contexts.

  Value Theory-Based Valuation: There are works discussing valuations for policies applied to the problem of trading goods and services and service valuation, but not, to the best of our knowledge, in the domain of compliance. [EK07], building on [LAS+06, SL05, LS06], reports on a method for valuating policies using several alternative valuation models including discrete and continuous functions, piece-wise linear functions, and pattern-based functions.

- **Rules & Verification Dimension:**

  Visual Pattern-Based Business Rules: Using a business vocabulary for the business domain would allow users to model rules which are semantically closer to their domain of expertise. We think that combining a rich business vocabulary with rule patterns (temporal and structural), as in [Ama12, XLW08, YMH+06] which build on the property specification patterns in [MGJ98, GL06] is a very promising perspective.

Temporal Logic Language: Although we support temporal specifications in a limited way, it is worthwhile exploring using a pure temporal logic language for rule modeling such as CTL/LTL [BK08]. This will require us to use a different target semantic domain than Alloy, e.g., labelled transition systems or petri nets.

- **Process Dimension:**

Rich Process State Model: Up to this stage, we restricted ourselves to only two basic states of a process task. It is our intuition that a more complete process state model description would increase the applicability of our approach by increasing the scope of process executions it can deal with, e.g., by considering task states such as assigning, delegating or cancelling process tasks [Gaa10].

- **Language Dimension:**

Concrete Syntax Layer on the Alloy Analyzer: As stated in the limitations of our approach, using the raw output produced by the analyzer for debugging our Alloy formalization was a time-consuming and arduous task. The reason is that the output of the analyzer is always a set of atoms and relations, which obviously does not help to reason on domain concepts, such as policy and process. This is why we propose to develop a flexible mechanism allowing to define a concrete syntax for Alloy instance models.

10.5 Conclusion

In this thesis, we have contributed the CoReL method and stack of languages for modeling and enforcing compliance. Our vision was to provide a full-blown model-driven framework for modeling compliance requirements of various kinds to accurately represent the decision-making requirements defined by a regulation.

The framework we have built is based on a formalization of decision-making based on the policy paradigm. Whether this paradigm is the most adequate for the purpose of RCM is not a proven fact, although we argued for its inception from observing the nature of the RCM problem.

The visual concrete syntax of the CoReL language may be open for discussion. This is because it is motivated by the research objective of making it easier for business users to create policy representations of existing regulations. However, no empirical evidence, large-scale or not, has yet shown us that this is indeed the case. We can also say that empirical validation is required to test the following hypotheses: (i) graphical notation for policy modeling is useful, (ii) the ontological model underlying our language is adequate to the problem and the user (as a conceptual bridge), (iii) the notation defined for CoReL is intuitive to the business user.

Moreover, the formal verification techniques applied here each show some strengths and weaknesses. The Alloy model checking is bounded to reasonable sizes of state spaces, and uses the small-scope hypothesis. The other forms we mentioned such as model checking (e.g. Alpina [BHMR10, smc10], Lola [Awa10], NuSMV [CCG+02], Spin [Hol03]) present the major strength of high expressiveness but suffer from the problem pertaining to model-checking, i.e., state explosion.

The CoReL framework still needs much work in order to fulfill all the requirements elicited in the state of the art chapter. For example the way CoReL deals with Violations is still very basic. We do support neither violation localization nor explanation. Some extension points of CoReL, in particular the Compensation and Reparation constructs in CoReL, have not been illustrated in the implementation. As such, the full strength of CoReL's modeling elements has not been shown.

Another element is the rule modeling languages supported by CoReL, which are core to the evaluation of policy models. We have refrained from designing a new rule language since this is not where the challenge we identified lies. However, reusing the designed enterprise modeling language for defining a rich and expressive rule language, which is more accessible to business users would be an undeniable positive factor in the usability of CoReL. This is all the more true, since one of the non-trivial steps identified in using CoReL is the search and retrieval of rules in policies.

Appendix A

Publications

A.1

Appendix I-Publications Produced by the Author

Several peer-reviewed publications have been produced by the work that led to this thesis. Due to the multi-disciplinary nature of the research, these touch on a variety of topics such as business process management, semantic web (esp. Ontology engineering), business policy and rule management, requirements engineering, model driven engineering, web services and enterprise modeling.

I must reiterate my gratitude to all my very esteemed co-authors, as well as the members of the scientific community who helped produce these papers through their valuable reviews. The following list gives these publications in reverse chronological order:

- Marwane El Kharbili. Applying CoReL to the Healthcare HIPAA Privacy in Act. Sixth International Workshop on Requirements Engineering and Law. Special Track on Convergent Challenges in Legal Requirements Analysis and Modeling. In conjunction with the 21st International requirements Engineering Conference. 2013.
- Silvano Colombo Tosatto, Marwane El Kharbili, Guido Governatori, Pierre Kelsen, Qin Ma, and Leender van der Torre. Algorithms for basic compliance problems. 2nd International Workshop on Engineering Safety and Security Systems, ESSS 2013.
- Silvano Colombo Tosatto, Marwane El Kharbili, Guido Governatori, Pierre Kelsen, Qin Ma, and Leender van der Torre. Algorithms for basic compliance problems. In Benelux conference on Artificial Intelligence. 2012.
- Marwane El Kharbili. Business Process Regulatory Compliance Management Solution Frameworks: A Comparative Evaluation. In A. Ghose and F. Ferrarotti, editors, Asia-Pacific Conference on Conceptual Modelling (APCCM 2012), volume 130 of CRPIT, pages 23-32, 2012.
- Marwane El Kharbili, Qin Ma, Pierre Kelsen, and Elke Pulvermueller. CoReL: Policy-Based and Model-Driven Regulatory Compliance Management. In Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC), number ISSN: 1541-7719, ISBN: 978-1-4577-0362-1, pages 247-256. IEEE Computer Society Press, 2011.
- Marwane El Kharbili, Qin Ma, Pierre Kelsen, and Elke Pulvermueller. Enterprise Regulatory Compliance Modeling Using CoReL: An Illustrative Example. In Proceedings of the 13th IEEE Conference on Commerce and Enterprise Computing (CEC), number ISSBN: 978-1-4577-1542-6, pages 185-190. IEEE Computer Society Press, 2011.
- Marwane El Kharbili and Elke Pulvermueller. Semantic Technologies for Business and Information Systems Engineering: Concepts and Applications, chapter 16: Semantic Policies for Modeling Regulatory Process Compliance, pages 311-336. IGI Global, 2011.

- Ken Decreus, Marwane El Kharbili, Geert Poels, and Elke Pulvermueller. Policy-Enabled Goal-Oriented Requirements Engineering for Semantic Business Process Management. International Journal of Intelligent Systems - special issue on Goal-driven Requirements Engineering - Jonathan Lee, Wen-Tin Lee (Eds.), 25(8):784-812, August 2010.

- Marwane El Kharbili, Tobias Keil. Bringing Agility to Business Process Management: Rules Deployment in an SOA. In Emerging Web Services Technology, volume III of Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 157-170. Springer.

- Andreas Rusnjak, Andreas Speck, Hristov Hristomir, El Kharbili Marwane. Managing the dynamics of e/m commerce with a hierarchical overlapping business-value framework. In Proceedings of the 24th IEEE Conference on Advanced Information Networking and Applications (AINA 2010), The 6th international Symposium on Web and Mobile Information Services (WAMIS 2010), Perth, Australia, 2010.

- Marwane El Kharbili, Elke Pulvermueller. Service contract compliance management. In Emerging Web Services Technology, volume III of Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 105-116. Springer, 2010.

- Sebastian Stein, C. Stamber, M. El Kharbili, and P. Rubach. Semantic business process management: A case study. In G. Mentzas and A. Friesen, editors, Semantic Enterprise Application Integration for Business Processes: Service-Oriented Frameworks, pages 228-250. IGI Global, September 2009.

- Marwane El Kharbili. Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, volume 2 Volumes, chapter XXIII. Business Rules Management for Business Processes: From Modeling to Deployment, pages 540 - 563. IGI Publishing, May 2009.

- Marwane El Kharbili and Elke Pulvermueller. A Semantic Framework for Compliance Management in Business Process Management. In Proceedings of the 2nd International Conference on Business Process and Service Computing (BPSC'09) as Part of Software, Agents and Services for Business, Research, and E-Sciences (SABRE), Lecture Notes in Informatics (LNI), pages 60-80. Gesellschaft fuer Informatik, GI, March 2009.

- Ken Decreus, Marwane El Kharbili, Geert Poels, and Elke Pulvermueller. Bridging Requirements Engineering and Business Process Management. In Juergen; Muench and Peter; Liggesmeyer, editors, Workshop Proceedings of the SE 2009 - REBPM, volume 150 of Lecture Notes in Informatics, pages 215-222, Kaiserslautern, Germany, March 2009. Gesellschaft fuer Informatik (GI). in Conjunction with the Software Engineering Conference 2009 (SE2009).

- Marwane El Kharbili. Semantic compliance management in business process management. In W. Abramowicz, L. Maciaszek, R. Kowalczyk, A. Speck, editor, Proceedings of the Business Process, Services Computing and Intelligent Service Management, GI, Lecture Notes in Informatics, Lecture Notes in Informatics (LNI), pages 235-236. GI, March 2009.

- Marwane El Kharbili and Nenad Stojanovic. Semantic event-based decision management in compliance management for business processes. In Proceedings of the Intelligent Event Processing Symposium, pages 35-40, Stanford, USA., March 2009. AAAI Spring Symposium 2009.

- Sebastian Stein, Yves Lauer, and Marwane El Kharbili. Using template analysis as background reading technique for requirements elicitation. In GI Software Engineering, volume 143 of LNI, pages 127-138, Kaiserslautern, Germany, March 2009. GI.

- Marwane El Kharbili and Piotr Stolarski. Building-up a reference generic regulation ontology: A bottom-up approach. In Witold Abramowicz and Dominik Flejter, editors, BIS 2009 International Workshops, volume Volume 37, Part 7 of Lecture Notes in Business Information Processing. Business Information Systems Workshops., pages 268-279, Poznan, Poland, April 27-29, 2009.

- Wolfgang Runte and Marwane El Kharbili. Constraint checking for business process management. In Ruediger Reischuk Stefan Fischer, Erik Maehle, editor, Proceedings of the Workshop on Business Process Modeling and Realization, GI Informatik 2009, volume P-154 of Lecture Notes in Informatics (LNI), pages 4093-4103, Luebeck, Germany., 2009. Gesellschaft fuer Informatik, Bonn, GI.

- Andreas Rusnjak, Marwane El Kharbili, Andreas Speck. On leveraging business processes to deal with critical success factors. In Ruediger Reischuk Stefan Fischer, Erik Maehle (eds) Proceedings of the Workshop on Business Process Modeling and Realization, GI Informatik 2009, volume P-154 of Lecture Notes in Informatics (LNI), pages 4052-4066, Luebeck, Germany., 2009. Gesellschaft fuer Informatik, Bonn, GI.

- Marwane El Kharbili and Tobias Keil. Bringing agility to business process management: Rules deployment in an soa. In Proceedings of the European Conference on Web Services - Industrial Track - ECOWS08, Dublin, Ireland, November 2008.

- Marwane El Kharbili, Ana Karla Alves de Medeiros, Sebastian Stein, and Wil M. P. van der Aalst. Business process compliance checking: Current state and future challenges. In Proceedings of the Conference on Business Information Systems - Modellierung betrieblicher Informationssysteme (MobIS), volume 141 of LNI, pages 107-113, Saarbruecken, Germany., November 2008.

- Sebastian Stein, Katja Barchewitz and Marwane El Kharbili. Enabling business experts to discover web services for business process automation. In Emerging Web Services Technology, volume III of Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 23-29. Springer, 2008.

- Marwane El Kharbili, Sebastian Stein, and Elke Pulvermueller. Policy-based semantic compliance checking for business process management. In Gemischter Workshop zur Referenzmodellierung und semantische Geschaeftsprozessmodellierung, volume 420 of CEUR Workshop Proceedings:, pages 178-192, Saarbruecken, Germany, November 2008.

- Sebastian Stein, Christian Stamber, Marwane El Kharbili, and Pavel Rubach. Semantic business process management: An empirical case study. In Gemischter Workshop zu Referenzmodellierung und semantische Geschaeftsprozessmodellierung, volume 420 of CEUR Workshop Proceedings, pages 165-177, Saarbruecken, Germany, 2008.

- Marwane El Kharbili and Elke Pulvermueller. Service contract compliance management. In Proceedings of the 3rd Workshop on Emerging Web Services Technology, pages 87-96, Dublin, Ireland, November 2008.

- Marwane El Kharbili, Sebastian Stein, Ivan Markovic, and Elke Pulvermueller. Towards policy-powered semantic enterprise compliance management - discussion paper. In Proceedings of the 3rd International Workshop on Semantic Business Process Management (SBPM), pages 16-21, Tenerife, Spain, Juni 2 2008. CEUR Workshop Proceedings.

- Sebastian Stein, Christian Stamber, and Marwane El Kharbili. ARIS for semantic business process management. In In Lecture Notes in Business Information Processing. Workshop on Advances in Semantics for Web Services (semantic4ws), volume 17, Part 7, pages 498-509, Milan, Italy, September 1, 2008 2008.

- Marwane El Kharbili and Sebastian Stein. Compliance management auf basis von semantischen richtlinien. In Klaus-Peter Faehnrich, Stefan Kuehne, and Maik Thraenert, editors, Model- Driven Integration Engineering, volume XI of Leipziger Beitraege zur Informatik XI, pages 253-262. Eigenverlag Leipziger Informatik-Verbund (LIV), Leipzig, Germany, September 2008.

- Marwane El Kharbili and Sebastian Stein. Compliance Management auf Basis von Semantischen Richtlinien. In Klaus-Peter Faehnrich, Stefan Kuehne, and Maik Thraenert, editors, Model- Driven Integration Engineering, volume XI of Leipziger Beitraege zur Informatik XI, pages 253-262. Eigenverlag Leipziger Informatik-Verbund (LIV), Leipzig, Germany, September 2008.

- Marwane El Kharbili, Sebastian Stein, Ivan Markovic, and Elke Pulvermueller. Towards a framework for semantic business process compliance management. In Sadiq Shazia, Indulska Marta, and zur Muehlen Michael, editors, Proceedings of the GRCIS workshop., volume 339 of CEUR Workshop Proceedings, pages 1-15, Montpellier, France, June 17th 2008.

- Christian Stamber, Sebastian Stein, and Marwane El Kharbili. Prototypical implementation of a pragmatic approach to semantic web service discovery during process execution. In Witold Abramowicz and Dieter Fensel, editors, 11th International Conference on Business Information Systems (BIS), volume 7 of LNBIP, pages 201-212, Innsbruck, Austria, May 2008. Springer.

- Sebastian Stein, Katja Barchewitz, and Marwane El Kharbili. Enabling business experts to discover web services for business process automation. In Cesare Pautasso and Thomas Gschwind, editors, 2nd Workshop on Emerging Web Services Technology, pages 19-35, Halle, Germany, November 2007.

Bibliography

[Aal05] Wil M. P. van der Aalst. Business alignment: using process mining as a tool for delta analysis and conformance testing. *Requirements Engineering*, 10:198–211, 2005.

[Abd10] Norris Syed Abdullah. Information systems research: Aligning to industry challenges in management of regulatory compliance. In *PACIS 2010 Proceedings*, 2010.

[ABJM07] Artur Boronat and José Meseguer. Algebraic Semantics of EMOF/OCL Metamodels. Technical report, Technical Report UIUCDCS-R-2007-2904, Computer Science Department, University of Illinois at Urbana-Champaign, USA, 2007.

[ACKP11] Wihem Arsac, Luca Compagna, Samuel Paul Kaluvuri, and Serena Elisa Ponta. Security validation tool for business processes. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, SACMAT '11, pages 143–144, New York, NY, USA, 2011. ACM.

[ACPP11] Wihem Arsac, Luca Compagna, Giancarlo Pellegrino, and Serena Ponta. Security validation of business processes via model-checking. In Úlfar Erlingsson, Roel Wieringa, and Nicola Zannone, editors, *Engineering Secure Software and Systems*, volume 6542 of *Lecture Notes in Computer Science*, pages 29–42. Springer Berlin / Heidelberg, 2011.

[ADO+08] Wil M. P. van der Aalst, Marlon Dumas, Chun Ouyang, Anne Rozinat, and Eric Verbeek. Conformance checking of service behavior. *ACM Trans. Internet Technol.*, 8:13:1–13:30, May 2008.

[ADW08] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic. In Marlon Dumas, Manfred Reichert, and Ming C. Shan, editors, *Proceedings of the 6th Int'l Conference on Business Process Management (BPM 2008)*, LNCS, pages 326–341, Milano, Italy, 2008. Springer Verlag.

[AFL99] Jim Alves Foss and Fong Shing Lam. Dynamic Denotational Semantics of Java. In *Formal Syntax and Semantics of Java*, pages 201–240, London, UK, 1999. Springer-Verlag.

[AGERPS08] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. Model-Driven Language Engineering: The ASMETA Case Study. In *Proceedings of the Third International Conference on Software Engineering Advances (ICSEA '08)*, pages 373–378, Washington, DC, USA, October 2008. IEEE Computer Society.

[AGTW11] Ahmed Awad, Rajeev Goré, James Thomson, and Matthias Weidlich. An iterative approach for business process template synthesis from compliance rules. In Haralambos Mouratidis and Colette Rolland, editors, *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 406–421. Springer Berlin / Heidelberg, 2011.

[AH02] Wil van der Aalst and Kees Max van Hee. *Workflow management : models, methods, and systems*, pages 359–363. Cooperative information systems. MIT Press, Cambridge, Mass., 2002.

[AHK07] J. Anon, F. H., and J.M. Kovatch. Integrating sarbanes-oxley controls into an investment firm governance framework. *The Journal of Investment Compliance*, 8(1):40–43, 2007.

[AIS09] Norris Syed Abdullah, Marta Indulska, and Sadiq Shazia. A study of compliance management in information systems research. In *ECIS 2009 Proceedings*, 2009.

[AK02] Colin Atkinson and Thomas Kuehne. The role of metamodeling in MDA. In Workshop in Software Model Engineering (WISME@UML), 2002.

[AK03] Colin Atkinson and Thomas Kuehne. Model-driven development: A metamodeling foundation. IEEE Software, 20(5):36–41, 2003.

[AK09] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley, Upper Saddle River, NJ, 2009.

[Ama12] Elgammal Amal. *Towards a Comprehensive Framework for Business Process Compliance*. PhD thesis, Tilburg University, 2012.

[aP13] Boolean Satisfiability Group at Princeton. zchaff. www.princeton.edu/~chaff/zchaff.html, 30 June 2013.

[ASI10] Norris Syed Abdullah, Shazia W. Sadiq, and Marta Indulska. Emerging challenges in information systems research for regulatory compliance management. In *CAiSE*, pages 251–265, 2010.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[ASW09] Ahmed Awad, Sergey Smirnov, and Mathias Weske. Resolution of compliance violation in business process models: A planning-based approach. In Robert Meersman, Tharam Dillon, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2009*, volume 5870 of *Lecture Notes in Computer Science*, pages 6–23. Springer Berlin / Heidelberg, 2009.

[Awa10] Ahmed Awad. *A Compliance Management Framework for Business Process Models*. PhD thesis, Hasso-Plattner Institute, 2010.

[AWW09] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Specification, verification and explanation of violation for data aware compliance rules. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *Service-Oriented Computing*, volume 5900 of *Lecture Notes in Computer Science*, pages 500–515. Springer Berlin / Heidelberg, 2009.

[AWW11] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Visually specifying compliance rules and explaining their violations for business processes. *Journal of Visual Languages & Computing*, 22(1):30 – 55, 2011. Special Issue on Visual Languages and Logic.

[BBB+08] M. Baldoni, C. Baroglio, I. Brunkhorst, N. Henze, E. Marengo, and V. Patti. Constraint modeling for curriculum planning and validation. *International Journal of Interactive Learning Environments*, 19(1):83–123, 2008.

[BCT05] Alan W. Brown, Jim Conallen, and Dave Tropeano. Introduction: Models, modeling, and model-driven architecture (mda). In Sami Beyeda, Matthias Book, and Volker Gruhn, editors, *Model-Driven Software Development*, pages 1–16. Springer-Verlag, 2005.

[BHMR10] Didier Buchs, Steve Hostettler, Alexis Marechal, and Matteo Risoldi. Alpina: A symbolic model checker. In Johan Lilius and Wojciech Penczek, editors, *Applications and Theory of Petri Nets*, volume 6128 of *Lecture Notes in Computer Science*, pages 287–296. Springer Berlin / Heidelberg, 2010.

[BIS+07] Wasana Bandara, Marta Indulska, Shazia Sadiq, Sandy Chong, Michael Rosemann, and Peter Green. Major issues in business process management: An expert perspective. Technical report, School of ITEE, University of Queensland, Australia, 2007.

[BK06] Borzoo Bonakdarpour and Sandeep S. Kulkarni. Towards reusing formal proofs for verification of fault-tolerance. In *AFM (Automated Formal Methods)*, 2006.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model-Checking*. The MIT Press, May 2008.

[BKB00] Arthur H. M. ter Hofstede Bartek Kiepuszewski and Christoph Bussler. On structured workflow modelling. In *In Proceedings of the 12th Inter- national Conference on Advanced Information Systems Engineering, CAiSE '00.*, pages 431–445. Springer-Verlag., 2000.

[BMG08] T. Butler and D. Mc Govern. Adopting it to manage compliance and risks: An institutional perspective. In Golden W., Acton T., Conboy K., van der Heijden H., and Tuunainen V. K., editors, *Proceedings of the 16th European Conference on Information Systems*, page 1043, 2008.

[BMPS10] Elisa Bertino, Lorenzo Martino, Federica Paci, and Anna Squicciarini. Access control for business processes. In *Security for Web Services and Service-Oriented Architectures*, pages 159–177. Springer Berlin Heidelberg, 2010.

[BMS10] Sabine Buckl, Florian Matthes, and Christian M. Schweda. A technique for annotating ea information models with goals. In Joseph Barjis, Wil M. P. van der Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Enterprise and Organizational Modeling and Simulation*, volume 63 of *Lecture Notes in Business Information Processing*, pages 113–127. Springer Berlin Heidelberg, 2010.

[Bro86] Frederick P. (Jr.) Brooks. No Silver Bullet – Essence and Accidents of Software Engineering (Invited Paper). In *IFIP 10th World Computer Congress*, pages 1069–1076, Dublin, Ireland, September 1986.

[CATK03] Colin Atkinson and Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20:36–41, September/October 2003.

[CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the Fourth ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, pages 238–252, New York, NY, USA, 1977. ACM.

[CC79] Patrick Cousot and Radhia Cousot. Systematic Design of Program Analysis Frameworks. In *Proceedings of the Sixth ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '79, pages 269–282, New York, NY, USA, 1979. ACM.

[CC00] Patrick Cousot and Radhia Cousot. Temporal Abstract Interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Massachussets, January 2000. ACM Press.

[CCG+02] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In Ed Brinksma and KimGuldstrand Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer Berlin Heidelberg, 2002.

[CCHP04] Ariane Chapelle, Yves Crama, Georges Hubner, and Jean-Philippe Peeters. Basel ii and operational risk: Implications for risk measurement and management in the financial sector. Research series 200405-7, National Bank of Belgium, May 2004.

[CDOP02] Gennaro Costagliola, Andrea Delucia, Sergio Orefice, and Giuseppe Polese. A classification framework to support the design of visual languages. *Journal of Visual Languages & Computing*, 13(6):573–600, 2002.

[CFPC09] Tilburg University COMPAS FP7 Project Consortium. D2.2-initial specification of compliance language constructs and operators. revision 2.0., 2009.

[CH03] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.

[CK08] Baier Christel and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, Cambridge, 2008.

[CKO92] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, 1992.

[Com08] Benoît Combemale. *Metamodeling Approach for Model Simulation and Verification — Application to Process Engineering*. PhD thesis, Institut National Polytechnique de Toulouse (INPT) / École Nationale Supérieure d'Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et de Télécommunications (ENS-EEEIHT), July 2008. (in french).

[Con09] Jan Fanta (ANECT) Master Consortium. Regulatory requirements analysis. Master Project - Deliverable D1.1.2, 24th of June 2009.

[Cou02] Patrick Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science*, 277(1–2):47–103, 2002.

[CRR09] Fernando Castor Filho, Alexander Romanovsky, and Cecília M. F. Rubira. Improving reliability of cooperative concurrent systems with exception flow analysis. *J. Syst. Softw.*, 82(5):874–890, 2009.

[CTEKG+13] Silvano Colombo Tosatto, Marwane El Kharbili, Guido Governatori, Pierre Kelsen, Qin Ma, and Leender van der Torre. Algorithms for basic compliance problems. In *International Workshop on Engineering Safety and Security Systems (ESSS)*. 2013.

[CTGKvdT12] Silvano Colombo Tosatto, Guido Governatori, Pierre Kelsen, and Leendert van der Torre. Business process compliance is hard. Technical report, NICTA, 2012.

[CW09] Anne Cleven and Robert Winter. Regulatory compliance in information systems research - literature analysis and research agenda. In *BMMDS/EMMSAD Proceedings*, pages 174–186, 2009.

[Dav93] Thomas H. Davenport. Process innovation: reengineering work through information technology. Harvard Business School Press, Boston, MA, USA., 1993.

[DB07] Rob Davis and Eric Brabänder. *ARIS Design Platform: Getting Started with BPM*. Springer, 1st edition, June 2007.

[DDO+11] Schumm David, Karastoyanova Dimka, Kopp Oliver, Leymann Frank, Sonntag Mirko, and Strauch Steve. Process fragment libraries for easier and faster development of process-based applications. *Journal of Systems Integration*, 2(1):39 – 55, 2011.

[DH08] Jan L. G. Dietz and Jan A. P. Hoogervorst. Enterprise ontology in enterprise engineering. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 572–579, New York, NY, USA, 2008. ACM.

[dLV04] Juan de Lara and Hans Vangheluwe. Defining Visual Notations and their Manipulation Through Meta-Modelling and Graph Transformation. *Journal of Visual Languages & Computing*, 15(3 – 4):309 – 330, August 2004.

[DNS08] Nirmit Desai, Nanjangud C. Narendra, and Munindar P. Singh. Checking correctness of business contracts via commitments. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 787–794, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[DON+10] Schumm David, Tueretken Oktay, Kokash Natallia, Elgammal Amal, Leymann Frank, and van den Heuvel Willem Jan. Business process compliance through reusable units of compliant processes. In Florian Daniel and Federico Michele Facca, editors, *ICWE Workshops - Proceedings of the 1st Workshop on Engineering SOA and the Web (ESW'10)*, volume 6385 of *Lecture Notes in Computer Science*, pages 325–337, Vienna, Austria, 26 May 2010 2010. Springer.

[DvdAtH05] Marlon Dumas, Wil M. van der Aalst, and Arthur H. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley-Interscience, 2005.

[DY13] Greg Dennis and Kuat Yessenov. Forge. bounded program verification. sdg.csail.mit.edu/forge/, June 2013.

[ea09a] Alberto Sanna et al. Stakeholder requirements analysis. Master Project - Deliverable D1.1.2, 2009.

[ea09b] Marc Lankhorst et al. *Enterprise Architecture at Work.* The Enterprise Engineering. Springer Berlin / Heidelberg, second edition, 2009.

[ECCB11] Yehia Elrakaiby, Frédéric Cuppens, and Nora Cuppens-Boulahia. Formal enforcement and management of obligation policies. *Data & Knowledge Engineering*, 2011.

[EK07] Marwane El Kharbili. Policy-based collaboration in semantic business processes. Diplomarbeit - fakultaet fuer informatik., Universitaet Karlsruhe (Karlsruhe Institute of Technology - KIT.edu), Karlsruhe, Germany., 2007. Master Thesis (Diplomarbeit).

[EKAdMSvdA08] M. El Kharbili, A. K. Alves de Medeiros, S. Stein, and W. M. P. van der Aalst. Business process compliance checking: Current state and future challenges. In *Proceedings of the Conference on Business Information Systems - Modellierung betrieblicher Informationssysteme (MobIS)*, volume 141 of *LNI*, pages 107–113, Saarbruecken, Germany., November 2008.

[EKP09] Marwane El Kharbili and Elke Pulvermueller. A Semantic Framework for Compliance Management in Business Process Management. In *Proceedings of the 2nd International Conference on Business Process and Service Computing (BPSC'09) as Part of Software, Agents and Services for Business, Research, and E-Sciences (SABRE)*, Lecture Notes in Informatics (LNI), pages 60–80. Gesellschaft fuer Informatik, GI, 2009.

[EKSMP08] M. El Kharbili, S. Stein, I. Markovic, and E. Pulvermueller. Towards a framework for semantic business process compliance management. In Sadiq Shazia, Indulska Marta, and zur Muehlen Michael, editors, *Proceedings of the GRCIS workshop.*, volume 339 of *CEUR Workshop Proceedings*, pages 1–15, Montpellier, France, June 17th 2008.

[ES03] Ed Seidewitz. What Models Mean. *IEEE Software*, 20:26–32, September 2003.

[ES09] M. El Kharbili and N. Stojanovic. Semantic event-based decision management in compliance management for business processes. In *Proceedings of the Intelligent Event Processing Symposium*, pages 35–40, Stanford, USA., March 2009. AAAI Spring Symposium 2009.

[ES13] Niklas Een and Niklas Sorensson. The minisat page. `minisat.se`, 2013.

[ETvdHP10a] A. Elgammal, O. Turetken, W. van den Heuvel, and M. Papazoglou. On the formal specification of regulatory compliance: A comparative analysis. In Maglio et al., editor, *Proceedings International Performance Assessment and Auditing in Service Computing Workshop.* ICSOC10 workshops, Springer, USA, 2010 2010.

[ETvdHP10b] A. Elgammal, O. Turetken, W. van den Heuvel, and M. Papazoglou. Root-cause analysis of design-time compliance violations on the basis of property patterns. In Maglio et al., editor, *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC10)*, pages pp. 17–31, USA, 2010. Springer.

[FES05] Alexander Förster, Gregor Engels, and Tim Schattkowsky. Activity diagram patterns for modeling quality constraints in business processes. In *MoDELS*, pages 2–16, 2005.

[FESS06] Alexander Förster, Gregor Engels, Tim Schattkowsky, and Ragnhild Van Der Straeten. A pattern-driven development process for quality standard-conforming business process models. In *VL/HCC*, pages 135–142, 2006.

[FESS07] Alexander Förster, Gregor Engels, Tim Schattkowsky, and Ragnhild Van Der Straeten. Verification of business process quality constraints based on visual process patterns. In *TASE*, pages 197–208, 2007.

[FMPT01] Ariel Fuxman, John Mylopoulos, Marco Pistore, and Paolo Traverso. Model checking early requirements specifications in tropos. In *Fifth IEEE International Symposium on Requirements Engineering*, pages 174–181. IEEE Computer Society, 2001.

[fMS13] Centers for Medicare and Medicaid Services. HIPAA regulation, 2013.

[FPR07] F.Yip, N. Parameswaran, and P. Ray. Rules and ontology in compliance management. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, number 1541-7719, page 435, Washington, DC, USA, 2007. Washington, DC, USA, IEEE Computer Society.

[FR07] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[Gaa10] Khaled Gaaloul. *Une Approche Securisee pour la Delegation Dynamique de Taches dans les systems de Gestion de Workflow*. PhD thesis, Universite Henri Poincare, Nancy I, 2010.

[Gal13] Juan Pablo Galeoetti. Taco: Translation of annotated code. `www.dc.uba.ar/inv/grupos/fm_folder/TACO`, June 2013.

[GJH03] Gerard J. Holzmann. *The* Spin *Model Checker: Primer and Reference Manual*. Addison-Wesley, September 2003.

[GK07] A.K. Ghose and G. Koliadis. Auditing business process compliance. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC-2007)*, volume 4749 of *Lecture Notes in Computing Sicence*, pages 169–180, 2007.

[GK14] Loiïc Gammaitoni and Pierre Kelsen. Domain-specific visualization of alloy instances. In *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 324–327. Springer, 2014.

[GL06] Volker Gruhn and Ralf Laue. Patterns for timed property specifications. *Electronic Notes in Theoretical Computer Science*, 153(2):117 – 133, 2006. Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005).

[GLM+05] Christopher Giblin, Alice Y. Liu, Samuel Müller, Birgit Pfitzmann, Xin Zhou. M. f. Moens, and P. Spyns (Eds). Regulations expressed as logical models (realm). In *Proceedings of the Conference on Legal Knowledge and Information Systems (JURIX'2005) - The Eighteenth Annual Conference*, pages 37–48, 2005.

[GLR+02] Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel, and Andrew Wood. Transformation: The missing link of. In *ICGT '02: Proceedings of the First International Conference on Graph Transformation*, pages 90–105, London, UK, 2002. Springer-Verlag.

[GM05] Guido Governatori and Zoran Milosevic. Dealing with contract violations: formalism and domain specific language. In *Proceedings of the Conference on Enterprise Computing EDOC 2005.*, pages 46–57. IEEE Press., 2005.

[GMS06] Guido Governatori, Zoran Milosevic, and Shazia Sadiq. Compliance checking between business processes and business contracts. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages pp. 221–232, 2006.

[Gov05] Guido Governatori. Representing business contracts in ruleml. *International Journal of Cooperative Information Systems*, 14(2-3):181–216, 2005.

[GR10a] Guido Governatori and Antonino Rotolo. A conceptually rich model of business process compliance. In *Proceedings of the 7th Asia-Pacific Conference on Conceptual Modelling (APCCM 2010), Brisbane, Australia*, 2010.

[GR10b] Guido Governatori and Antonino Rotolo. Norm compliance in business process modeling. In Mike Dean, John Hall, Antonino Rotolo, and Said Tabet, editors, *Semantic Web Rules*, volume 6403 of *Lecture Notes in Computer Science*, pages 194–209. Springer Berlin / Heidelberg, 2010.

[Gro08] OMG (Object Management Group). Omg business modeling specifications-semantics of business vocabulary and business rules (sbvr). version 1.0, 2008.

[Gro13] OMG (Object Management Group). Omg business modeling specifications-semantics of business vocabulary and business rules (sbvr). in process version., April 2011 2013.

[GRS09] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. A Semantic Framework for Metamodel-Based Languages. Ase, 16(3–4):415–454, December 2009.

[GRS10] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. Combining Formal Methods and Mde Techniques for Model-Driven System Design and Analysis. JAS, 3(1–2):1–18, 2010.

[GV06] Stijn Goedertier and Jan Vanthienen. *Designing Compliant Business Processes with Obligations and Permissions*, volume 4103 of *LNCS*, pages 5–14. Springer Verlag, 2006. S. Goedertier, J. Vanthienen (2006) Designing Compliant Business Processes with Obligations and Permissions. J eder, S. Dustdar et al. (Eds.) BPM 2006 Workshops, LNCS 4103, pp 5-14. Springer Verlag 2006.

[HC03] Michael Hammer and James Champy. *Reengineering the corporation : a manifesto for business revolution.* HarperBusiness Essentials, New York, 1st harperbusiness essentials pbk. edition, 2003. 2004295374 Michael Hammer & James Champy. ill. ; 21 cm. "A HarperBusiness book." "Revised and updated with a new author's note"–Cover. Rev. ed. of the c2001 ed. Includes index.

[Hen90] Matthew Hennessy. *The semantics of programming languages: an elementary introduction using structural operational semantics.* John Wiley & Sons, Inc., New York, NY, USA, 1990.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[Hol03] Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley, September 2003.

[Hoo09] Jan A. P. Hoogervorst. Enterprise governance. In *Enterprise Governance and Enterprise Engineering*, The Enterprise Engineering Series, pages 261–342. Springer Berlin Heidelberg, 2009.

[HR00] M. Huth and M. Ryan. Logic in computer science: Modelling and reasoning about systems. Cambridge University Press., 2000.

[HRNFN92] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: a Formal Introduction*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[HWG09] Jörg Hoffmann, Ingo Weber, and Guido Governatori. On compliance checking for clausal constraints in annotated process models. *Information Systems Frontiers*, pages 1–23, 2009.

[INR13] INRIA. The Coq proof assistant. http://coq.inria.fr/, July 2013.

[Jac02] Daniel Jackson. Micromodels of software: Lightweight modelling and analysis with alloy. Technical report, Software Design Group, MIT Lab for Computer Science, 2002.

[Jac12] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis. Revised Edition.* The MIT Press, 2012.

[Jac13] Daniel Jackson. Alloy case studies archive. alloy.mit.edu/alloy/citations/case-studies.html, June 2013.

[JB06a] Jean Bézivin. In Search of a Basic Principle for Model Driven Engineering. *Council of European Professional Informatics Societies (*Cepis*)* UpGrade *Journal*, V (Uml and Model Engineering)(2):21 – 24, April 2006.

[JB06b] Carol Rozwell John Bace. Understanding the components of compliance. Gartner Research - ID Number: G00137902, 7 July 2006.

[JMKH07] Kuester Jochen M., Ryndina Ksenia, and Gall Harald. *Generation of Business Process Models for Object Life Cycle Compliance*, volume 4714 of *Lecture Notes in Computer Science*, pages 165–181. Springer, Berlin / Heidelberg, 2007.

[JRAS97] John Rekers and Andy Schürr. Defining and Parsing Visual Languages with Layered Graph Grammars. *Journal of Visual Languages & Computing*, 8(1):27–55, February 1997.

[JW96] Daniel Jackson and Jeanette Wing. Lightweight formal methods. In Hossein Saiedian, editor, *oundtable contribution to: An invitation to formal methods*, volume 29, pages 16 – 30. IEEE Computer, 1996.

[JWAK03] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[KA09] Natallia Kokash and Farhad Arbab. Formal behavioral modeling and compliance analysis for service-oriented systems. In Frank de Boer, Marcello Bonsangue, and Eric Madelaine, editors, *Formal Methods for Components and Objects*, volume 5751 of *Lecture Notes in Computer Science*, pages 21–41. Springer Berlin / Heidelberg, 2009.

[Kag04] Lalana Kagal. *A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments.* Phd thesis, Faculty of the Graduate School of the University of Maryland, 2004.

[Ken02] Stuart Kent. Model Driven Engineering. In *IFM '02: Proceedings of the Third International Conference on Integrated Formal Methods*, pages 286–298, London, UK, May 2002. Springer-Verlag.

[KGS10] Aqueo Kamada, Guido Governatori, and Shazia Sadiq. Transformation of sbvr compliant business rules to executable fcl rules. In *RuleML 2010: 4th International Web Rule Symposium*, number 6403, pages 153–161. Springer, 2010.

[Kha12] Marwane El Kharbili. Business process regulatory compliance management solution frameworks: A comparative evaluation. In A. Ghose and F. Ferrarotti, editors, *Asia-Pacific Conference on Conceptual Modelling (APCCM 2012)*, volume 130 of *CRPIT*, pages 23–32, Melbourne, Australia, 2012. ACS.

[KKdV10] N. Kokash, C. Krause, and E. P. de Vink. Data-aware design and verification of service compositions with reo and mcrl2. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2406–2413, New York, NY, USA, 2010. ACM.

[KKPP10] Decreus Ken, Marwane El Kharbili, Geert Poels, and Elke Pulvermueller. Policy-enabled goal-oriented requirements engineering for semantic business process management. *International Journal of Intelligent Systems - special issue on Goal-driven Requirements Engineering - Jonathan Lee, Wen-Tin Lee (Eds.)*, 25(8):784–812, August 2010.

[KL08] Akhil Kumar and Rong Liu. A rule-based framework using role patterns for business process compliance. In *RuleML*, pages 58–72, 2008.

[Kle08] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Number 978-0321553454. Addison-Wesley Professional, 2008.

[KM08] Pierre Kelsen and Qin Ma. A lightweight approach for defining the formal semantics of a modeling language. In *ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MODELS 2008)*, volume 5301 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008.

[KMKP11a] Marwane El Kharbili, Qin Ma, Pierre Kelsen, and Elke Pulvermueller. CoReL: Policy-based and model-driven regulatory compliance management. In *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, number ISSN: 1541-7719, ISBN: 978-1-4577-0362-1, pages 247–256. IEEE Computer Society Press, 2011.

[KMKP11b] Marwane El Kharbili, Qin Ma, Pierre Kelsen, and Elke Pulvermueller. Enterprise regulatory compliance modeling using corel: An illustrative example. In *Proceedings of the 13th IEEE Conference on Commerce and Enterprise Computing (CEC)*, number ISSBN: 978-1-4577-1542-6, pages 185–190. IEEE Computer Society Press, 2011.

[KNS92] G. Keller, M. Nuettgens, and A. W. Scheer. Semantische prozessmodellierung auf der grundlage ereignisgesteuerter prozessketten (epk). Technical Report Heft 89, Universitaet des Saarlandes, Saarbruecken, Germany, 1992.

[KS09] Marwane El Kharbili and Piotr Stolarski. Building-up a reference generic regulation ontology: A bottom-up approach. In Witold Abramowicz and Dominik Flejter, editors, *BIS 2009 International Workshops*, volume 37, Part 7 of *Lecture Notes in Business Information Processing. Business Information Systems Workshops*, pages 268–279, 2009.

[KT98] G. Keller and T. Teufel. Sap r/3 process oriented implementation: Iterative process prototyping. Addison-Wesley, 1998.

[KT08] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.

[KWB03] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[LAS+06] Steffen Lamperter, Anupriya Ankolekar, Rudi Studer, Daniel Oberle, and Christof Weinhardt. A policy framework for trading configurable goods and services in open electronic markets. In *Proceedings of the 8th International Conference on Electronic Commerce: The new E-Commerce: Innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet*, pages 162–173, 2006.

[Lev02] B. Lev. Where have all of enron's intangibles gone? *Journal of Accounting and Public Policy*, 21:131–135, 2002.

[LGKWNY08] Lars Grunske, Kirsten Winter, and Nisansala Yatapanage. Defining the Abstract Syntax of Visual Languages with Advanced Graph Grammars–A Case Study Based on Behavior Trees. *Journal of Visual Languages & Computing*, 19(3):343 – 379, June 2008.

[LGRMD08] Linh Thao Ly, Kevin Goeser, Stefanie Rinderle-Ma, and Peter Dadam. Compliance of semantic constraints: A requirements analysis for process management systems. In Shazia Sadiq, Marta Indulska, and Michael zur Muehlen, editors, *Proceedings of the workshop on Governance, Risk and Compliance for Information Systems (GRCIS 2008)*, volume 339, pages 31–45, Montpellier, France, June 17th 2008. CEUR Workshop Proceedings.

[Lin08] Yun Lin. *Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability.* PhD thesis, Norwegian University of Science and Technology., Trondheim, Norway, 2008.

[LLK09] Stephen S.G. Lee, Eng Wah Lee, and Ryan K. L. Ko. Business process management (bpm) standards: a survey. *Business Process Management Journal*, 15(5):744–791, 2009.

[LMX07] Y. Liu, S. Mueller, and K. Xu. A static compliance-checking framework for business process models. *IBM Syst. J.*, 46(2):335–361, 2007.

[LRMGD09] Linh Ly, Stefanie Rinderle-Ma, Kevin Göser, and Peter Dadam. On enabling integrated process compliance with semantic constraints in process management systems. *Information Systems Frontiers*, pages 1–25, 2009. 10.1007/s10796-009-9185-9.

[LS06] Steffen Lamperter and Bjoern Schnizler. Trading services in ontology-driven markets. In *Proceedings of the SAC'06, Dijon, France.* Information Management and Systems, University of Karlsruhe (TH), Karlsruhe, Germany, 2006.

[LSG07] Ruopeng Lu, Shazia Sadiq, and Guido Governatori. Compliance aware business process design. In *Proceedings of the 3rd International Workshop on Business Process Design (BPD'07)*, 2007.

[LSG08] R. Lu, S. Sadiq, and G. Governatori. Measurement of compliance distance in business work practice. *Information Systems Management*, (25):344–355, 2008.

[Ly13] Linh Thao Ly. *SeaFlows - A Compliance Checking Frameworkfor Supporting the Process Lifecycle.* University of Ulm., 2013.

[Mar10] Ivan Markovic. *Semantic Business Process Modeling.* PhD thesis, Karlsruher Institut für Technologie, 2010.

[MB02] L. Mikhailov and M. Butler. Combining b and alloy. In ZB 2002: Formal Specification and Development in Z and B, volume 2272 of Springer Lecture Notes, Grenoble, France., 2002.

[MEKEP11] Marwane El Kharbili and Elke Pulvermueller. *Semantic Technologies for Business and Information Systems Engineering: Concepts and Applications - Chapter 16: Semantic Policies for Modeling Regulatory Process Compliance*, pages 311–336. IGI Global, 2011.

[MGJ98] B. Dwyer Matthew, S. Avrunin George, and C. Corbett James. Property specification patterns for finite-state verification. In *Proceedings of the 2nd Workshop on Formal Methods in Software Practice*, 1998.

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4):316 – 344, December 2005.

[MJ02] Stephen J. Mellor and Marc J.Balcer. Executable uml: A foundation for model- driven architecture. Addison-Wesley Professional, 2002.

[MO07] John Mullins and Raveca Oarga. Model checking of extended ocl constraints on uml models in socle. In *Proceedings of the 9th IFIP WG 6.1 international conference on Formal methods for open object-based distributed systems*, FMOODS'07, pages 59–75, Berlin, Heidelberg, 2007. Springer-Verlag.

[Moo03] Daniel Moody. Comparative evaluation of large data model representation methods: The analyst's perspective. In Stefano Spaccapietra, Salvatore March, and Yahiko Kambayashi, editors, *Conceptual Modeling — ER 2002*, volume 2503 of *Lecture Notes in Computer Science*, pages 214–231. Springer Berlin / Heidelberg, 2003.

[Moo09] D. Moody. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on*, 35(6):756 –779, nov.-dec. 2009.

[MPvdAP08] Nataliya Mulyar, Maja Pesic, Wil van der Aalst, and Mor Peleg. Declarative and procedural approaches for modelling clinical guidelines: Addressing flexibility issues. In Arthur ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 335–346. Springer Berlin / Heidelberg, 2008.

[MV07] Farida Mostefaoui and Julie Vachon. Verification of aspect-uml models using alloy. In *Proceedings of the 10th international workshop on Aspect-oriented modeling*, AOM '07, pages 41–48, New York, NY, USA, 2007. ACM.

[MvH09] Daniel Moody and Jos van Hillegersberg. Evaluating the visual syntax of uml: An analysis of the cognitive effectiveness of the uml family of diagrams. In Dragan Gašević, Ralf Lämmel, and Eric Van Wyk, editors, *Software Language Engineering*, volume 5452 of *Lecture Notes in Computer Science*, pages 16–34. Springer Berlin / Heidelberg, 2009.

[MW09] Simha R. Magal and Jeffrey Work. *Essentials of Business Processes and Information Systems*. Wiley, 2009.

[NAAS05] Desai Nirmit, U. Mallya Ashok, K. Chopra Amit, and Munindar P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, vol. 31(no. 12):pp. 1015–1027, december 2005 2005. Nirmit Desai, Ashok U. Mallya, Amit K. Chopra, Munindar P. Singh, "Interaction Protocols as Design Abstractions for Business Processes," IEEE Transactions on Software Engineering, vol. 31, no. 12, pp. 1015-1027, Dec., 2005.

[Nam08] Kioumars Namiri. *Model-Driven Management of Internal Controls for Business Process Compliance*. PhD thesis, University of Karlsruhe, 2008.

[NS07a] Kioumars Namiri and Nenad Stojanovic. Applying semantics to sarbanes oxley internal controls compliance. 2007.

[NS07b] Kioumars Namiri and Nenad Stojanovic. A formal approach for internal controls compliance in business processes. In *8th Workshop on Business Process Modeling, Development, and Support (BPMDS07)*, page 9, Trondheim, Norway, 2007.

[NS07c] Kioumars Namiri and Nenad Stojanovic. *Pattern-Based Design and Validation of Business Process Compliance*, pages 59–76. 2007.

[OA07] Paul N. Otto and Annie I. Antón. Addressing legal requirements in requirements engineering. In *Proceedings of the Requirements Engineering Conference*, pages 5–14, 2007.

[OAvdHWM11] Turetken O., Elgammal A., van den Heuvel W., and Papazoglou M. Enforcing compliance on business processes through the use of patterns. In *Proceedings of the 19th European Conference on Information Systems (ECIS 2011)*, 2011.

[Obj03] Object Management Group. MDA Guide (version 1.0.1). Technical report, Object Management Group, June 2003.

[Obj06] Object Management Group. Meta-Object Facility 2.0 Core Specification (06-01-01). Technical report, Object Management Group, 2006.

[Obj10] Object Management Group. Object Constraint Language (OCL) Specification (Version 2.2, formal/2010-02-01). Technical report, Object Management Group, 2010.

[Obj11] Object Management Group. OMG / Unified Modeling Language (UML). Technical report, Object Management Group, 2011.

[OMG06] OMG. Object constraint language. OMG Available Specification, Version 2.0, May 2006.

[OMG11] OMG. Business process modeling notation (bpmn) specification. Technical report, Object Management Group (OMG), February 2011. http://www.omg.org/spec/BPMN/2.0/PDF.

[Org12] International Standards Organization. ISO/IEC 27000:2009 - information security management systems - overview and vocabulary, 2012.

[otUS02] Congress of the United States. Public company accounting reform and investor protection act (sarbanes-oxley act). Pub. L. No. 107-204, 116 Stat. 745, 2002.

[OZD08] Ernst Oberortner, Uwe Zdun, and Schahram Dustdar. Domain-specific languages for service-oriented architectures: An explorative study. In Petri Mähönen, Klaus Pohl, and Thierry Priol, editors, *ServiceWave*, volume 5377 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2008.

[PAN04] Bonatti Piero A. and Shahmehri et al. Nahid. Rule-based policy specification: State of the art and future work - rewerse project i2-d1 deliverable. 4 September 2004.

[PC81] Patrick Cousot. Semantic Foundations of Program Analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[Pes08] Maja Pesic. *Constraint-based workflow management systems: Shifting controls to users. Proefschrift.* PhD thesis, Beta Research School for Operations Management and Logistics, Technische Universiteit Eindhoven., Eindhoven, 2008.

[Pet62] C. A. Petri. *Communication with Automata.* PhD thesis, Universität Bonn, Germany, 1962.

[PGBD12] Artem Polyvyanyy, Luciano Garcıa-Banuelos, and Marlon Dumas. Structuring acyclic process models. *Information Systems*, 6(37):518– 538, 2012.

[PGS+05] Vineet Padmanabhan, Guido Governatori, Shazia Sadiq, Robert M. Colomb, and Antonino. Rotolo. Process modelling: The deontic way. In Marcus Stumptner, Sven Hartmann, and Yasushi Kiyoki, editors, *The Third Asia Pacific Conference on Conceptual Modelling (APCCM 2006)*. Hobart, Tasmania, 16-19 December 2005.

[PHM+09] Tan Phan, Jun Han, Ingo Müller, Malinda Kapuruge, and Steven Versteeg. Soabse: An approach to realizing business-oriented security requirements with web service security policies. In *SOCA*, pages 1–10. IEEE, 2009.

[PSSvdA07] M. Pesic, M. Schonenberg, N. Sidorova, and W. van der Aalst. *Constraint-Based Workflow Models: Change Made Easy*, pages 77–94. 2007.

[PVS13] Pvs specification and verification system. http://pvs.csl.sri.com/index.shtml, July 2013.

[RCJL06] Agrawal Rakesh, Johnson Christopher, Kiernan Jerry, and Frank Leymann. Taming compliance with sarbanes-oxley internal controls using database technology. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, number 0-7695-2570-9, page 92, 2006.

[RCR07] Schmidt Rainer, Bartsch Christian, and Oberhauser Roy. Ontology-based representation of compliance requirements for service processes. In Hepp Martin, Hinkelmann Knut, Karagiannis Dimitris, Klein Ruediger, and Stojanovic (eds.) Nenad, editors, *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007) in conjunction with the 3rd European Semantic Web Conference (ESWC 2007)*, number ISSN 1613-0073, pages 28–39. Proceedings of CEUR Workshops, June 7 2007. Innsbruck, Austria.

[Rey10] Mark C. Reynolds. Lightweight modeling of java virtual machine security constraints. In Marc rappier, Uwe Gloesser, Sarfraz Khurshid, Rogine Laleau, and Steve Reeves, editors, *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 146–159. Springer Berlin Heidelberg, 2010.

[Riv10] José Eduardo Rivera. *On The Semantics of Real-Time Domain-Specific Modeling of Languages*. PhD thesis, University of Malaga (Spain), Ocotber 2010.

[RvdA06] A. Rozinat and W. M. P. van der Aalst. *Decision Mining in ProM*, volume 4102 of *LNCS*, pages 420–425. Springer Berlin / Heidelberg, 2006.

[SAB10] Seyyed M.A. Shah, Kyriakos Anastasakis, and Behzad Bordbar. From uml to alloy and back again. In Sudipto Ghosh, editor, *Models in Software Engineering*, volume 6002 of *Lecture Notes in Computer Science*, pages 158–171. Springer Berlin Heidelberg, 2010.

[SAD+10] Ghanavati S., Siena A., Amyot D., Susi A., and Perini A. Towards a framework for business process compliance. In *Proceedings of the International Workshop on Goal-Based Process Engineering (WGBP'10)*, 2010.

[SASI10] Norris Syed Abdullah, Shazia Sadiq, and Marta Indulska. Emerging Challenges in Information Systems Research for Regulatory Compliance Management. In Barbara Pernici, editor, *Advanced Information Systems Engineering*, volume 6051 of *Lecture Notes in Computer Science*, chapter 21, pages 251–265. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[Sch99] August-Wilhelm Scheer. *ARIS - Business Process Frameworks*. Springer, Berlin, Germany, 3rd edition, 1999.

[Sch00a] August-Wilhelm Scheer. *ARIS - Business Process Modeling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000.

[Sch00b] August-Wilhelm W. Scheer. *Aris-Business Process Modeling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000.

[Sch06] Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[SF04] Howard Smith and Peter Fingar. *Business Process Management (BPM): The Third Wave*. 2004.

[SG10] Shazia Sadiq and Guido Governatori. Managing regulatory compliance in business processes. In Jan Brocke and Michael (Eds.) Rosemann, editors, *Handbook on Business Process Management 2*, pages 159–175. 2010.

[SGN07] Shazia Sadiq, Guido Governatori, and Kioumars Namiri. *Modeling Control Objectives for Business Process Compliance*, volume 4714 of *Lecture Notes in Computer Science*, book: business process management Modeling Control Objectives for Business Process Compliance, pages 149–164. Springer Berlin / Heidelberg, 2007.

[SL05] Andreas Eberhart. Steffen Lamperter, Daniel Oberle. Approximating service utility from policies and value function patterns. In *Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 159–168. IEEE Computer Society, June 2005.

[SLM+10] David Schumm, Frank Leymann, Zhilei Ma, Thorsten Scheibler, and Steve Strauch. Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. In Schumann/Kolbe/Breitner/Frerichs, editor, *Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI'10), Göttingen, Germany, February 23-25, 2010*, pages 2125–2137, Göttingen, Februar 2010. Universitätsverlag Göttingen.

[SLS06] Andreas Schaad, Volkmar Lotz, and Karsten Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *SACMAT*, pages 139–149, 2006.

[SM02] Andreas Schaad and Jonathan D. Moffett. A framework for organisational control principles. In *ACSAC*, pages 229–238, 2002.

[smc10] AlPiNA: A symbolic model checker. Didier buchs, steve hostettler, alexis marechal, and matteo risoldi. In Johan Lilius and Wojciech Penczek, editors, *Applications and Theory of Petri Nets - 31st International Conference, PETRI NETS, Proceedings.*, volume 6128 of *Lecture Notes in Computer Science*, pages 287–296. Springer Berlin / Heidelberg, 2010.

[STA05] August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. *Process Aware Information Systems: Bridging People and Software Through Process Technology*, chapter Process modeling using event-driven process chains. Wiley Interscience, 2005.

[Tay11] Frederick W. Taylor. The principles of scientific management., 1911.

[TBO10] Slim Turki and Marija Bjekovic-Obradovic. Compliance in e-government service engineering: State-of-the-art. In Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Jean-Henry Morin, Jolita Ralyté, and Mehdi Snene, editors, *Exploring Services Science*, volume 53, 2010.

[TEvdHP12] O. Turetken, A. Elgammal, W.-J. van den Heuvel, and M.P. Papazoglou. Capturing compliance requirements: A pattern-based approach. *Software, IEEE*, 29(3):28 –36, may-june 2012.

[TFL07] E. Hartman Thomas, Foley, and Lardner LLP. The cost of being public in the era of sarbanes-oxley, August 2 2007.

[TK06] Thomas Kühne. Matters of (Meta-) Modeling. *Software and Systems Modeling (*SoSyM*)*, 5:369–385, July 2006.

[TMA06] Breaux T.D., Vail M.W., and Anton A.I. Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In *Proceedings of Requirements Engineering, 14th IEEE International Conference*, number ISSN: 1090-705X, pages 49 – 58, 2006.

[tow07] A Framework to Formalise the MDE Foundations. In *International Workshop on Towers of Models (TOWERS)*, pages 14–30, Zurich, June 2007.

[UC13] Artois University and CNRS. The boolean satisfaction an optimization library in java. www.sat4j.org, 30 June 2013.

[Uni08] COMPAS FP7 Project Consortium Tilburg University. State-of-the-art in the field of compliance languages. Deliverable D2.1, http://www.compas-ict.eu/results.php, 2008.

[vdA06] Wil M.P. van der Aalst. Matching observed behavior and modeled behavior: An approach based on petri nets and integer programming. *Decision Support Systems-Elsevier*, 42(3):1843–1859, May 2006.

[vdAdBvD05] W.M.P. van der Aalst, H.T. de Beer, and B.F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer Berlin / Heidelberg, 2005.

[vdAtHKB03] Wil M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14:5–51, July 2003.

[vDdMV+05] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. *Applications and Theory of Petri Nets 2005*, volume 3536/2005 of *Lecture Notes in Computer Science*, chapter The ProM Framework: A New Era in Process Mining Tool Support, pages 444–454. Springer Berlin / Heidelberg, 2005.

[vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: an Annotated Bibliography. *SIGPLAN Not.*, 35(6):26–36, June 2000.

[vtKB03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[Web09] Ingo Weber. *Semantic Methods for Execution Level Business Modeling.* PhD thesis, Universitaet Karlsruhe (KIT), 2009.

[Wes07] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer, 1 edition, November 2007.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction (Foundations of Computing).* MIT Press, Cambridge, MA (USA), February 1993.

[WM10] Christian Wolter and Christoph Meinel. An approach to capture authorisation requirements in business processes. *Requir. Eng.*, 15(4):359–373, 2010.

[Wor99] Workflow Management Coalition. Terminology & Glossary. Document Number WFMC-TC-1011. Issue 3.0, 1999. Available from: http://www.wfmc.org/Download-document/WFMC-TC-1011-Ver-3-Terminology-and-Glossary-English.html.

[Wor11] Ellen Messmer (Network World). Cost of regulatory security compliance? - on average, 3.5 m$ - quarter of companies don't do annual internal it security compliance audits, January 31, 2011.

[WPD+11] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, Jan Mendling, and Mathias Weske. Process compliance analysis based on behavioural profiles. *Information Systems*, 36(7):1009–1025, 2011.

[WPDM10] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, and Jan Mendling. Process compliance measurement based on behavioural profiles. In Barbara Pernici, editor, *Advanced Information Systems Engineering*, volume 6051 of *Lecture Notes in Computer Science*, pages 499–514. Springer Berlin / Heidelberg, 2010.

[WSM08] Christian Wolter, Andreas Schaad, and Christoph Meinel. Task-based entailment constraints for basic workflow patterns. In *SACMAT*, pages 51–60, 2008.

[XLW08] Ke Xu, Ying Liu, and Cheng Wu. Bpsl modeler - visual notation language for intuitive business property reasoning. *Electronic Notes in Theoretical Computer Science*, 211:211 – 220, 2008. Proceedings of the Fifth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2006).

[YMH+06] Jian Yu, Tan Manh, Jun Han, Yan Jin, Yanbo Han, and Jianwu Wang. Pattern based property specification and verification for service composition. In Karl Aberer, Zhiyong Peng, Elke Rundensteiner, Yanchun Zhang, and Xuhui Li, editors, *Web Information Systems (WISE)*, volume 4255 of *Lecture Notes in Computer Science*, pages 156–168. Springer Berlin / Heidelberg, 2006.

[zM04] Michael zur Muehlen. *Workflow-Based Process Controlling*, volume 6 of *Advances in Information Systems and Management Science*. Logos Verlag Berlin, 2004.

[ZMR08] Michael Zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *Advanced information systems engineering*, pages 465–479. Springer Berlin Heidelberg, 2008.

[ZX04] Yingzhou Zhang and Baowen Xu. A Survey of Semantic Description Frameworks for Programming Languages. *ACM SIGPLAN Notices*, 39(3):14–30, March 2004.