WEB RULES TO INTERCHANGE POLICIES

by

Nima Kaviani
B.Sc., Bu-Ali Sina University, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Interactive Arts and Technology

© Nima Kaviani 2007
SIMON FRASER UNIVERSITY
Summer 2007

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.
**APPROVAL**

<table>
<thead>
<tr>
<th>Name</th>
<th>Nima Kaviani</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree</td>
<td>Master of Science</td>
</tr>
<tr>
<td>Title of thesis</td>
<td>Web Rules to Interchange Policies</td>
</tr>
<tr>
<td>Examining Committee:</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Dr. Christopher Shaw</td>
<td>Chair</td>
</tr>
<tr>
<td>Dr. Marek Hatala, Simon Fraser University</td>
<td>Senior Supervisor</td>
</tr>
<tr>
<td>Dr. Dragan Gašević, Athabasca University</td>
<td>Supervisor</td>
</tr>
<tr>
<td>Dr. Konstantin Beznosov, University of British Columbia</td>
<td>External Examiner</td>
</tr>
</tbody>
</table>

Date Approved: [Signature]
Declaration of Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website <www.lib.sfu.ca> at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada

Revised: Summer 2007
Abstract

Web rule languages with supports for different types of rules have recently emerged to make interaction between parties with different policies and business rules possible. The chance of describing the resources and users of a domain through the use of vocabularies is another feature supported by Web rule languages. Combination of these two properties makes Web rule languages an appropriate medium to make a hybrid model of representing both context and rules of a policy-aware system. This dissertation demonstrates a potential solution to the problem of exchanging information between different registries and policy-aware systems (especially Web resources) by defining an interchange framework to transform business rules and concepts from one language to another using a third intermediary language called REWERSE Rule Markup Language (R2ML).

Keywords:

Policy Languages, Web Rule Languages, Description Logic, Declarative Logic, Web Services, Rule Interchange Format, Semantic Web.
"To My Parents"

— for their support and encouragement throughout my entire life.
Acknowledgments

I am honoured and delighted to acknowledge the many people whose counsel, support, and encouragement have contributed immensely to the completion of my thesis.

First and foremost, I would like to express my deep gratitudes to my senior supervisor, Dr. Marek Hatala, whose constant encouragement played the most important role in keeping me on the track of my studies. When lots of pressure, as a consequence of migrating to a new country, had made it almost impossible for me to keep up with my studies, his constant support and understanding of the situation were the main incentives for me to keep going.

My sincere appreciations also go to Dr. Dragan Gašević for his constant guidance, feedback, and fortitude during my studies. His motivating attitudes and his extreme patience in baring with my interruptions to his own work, and also his openness to discuss my crude research ideas, gave me the great opportunity to fully understand and grasp the research ideas.

I would also like to thank Dr. Gerd Wagner and Dr. Adrian Guirca for being supportive and open to discuss the research issues regarding R2ML.

Finally, I would like to thank my great friends and lab mates, Ty Mey Eap (Timmy) and Shilpi Rao, and Kirsten Johnson for baring with me and providing a great research and social atmosphere at school.
Contents

Approval ii
Abstract iii
Quotation iv
Acknowledgments v
Contents vi
List of Tables x
List of Figures xi
List of Programs xv

1 Introduction 1
1.1 Background ........................................ 1
1.2 Motivations ......................................... 3
1.3 Scope of Research .................................. 5
1.4 The Research Question ............................. 6
1.5 The Structure of the Dissertation .................. 6

2 Policy Management and Web Service Technologies 8
2.1 Semantic Web ....................................... 9
  2.1.1 Semantic Web Ontologies ....................... 11
  2.1.2 Semantic Web Ontology Languages ............. 12
2.2 Semantic Web Services ................................................. 14
  2.2.1 OWL-S ............................................................... 16
  2.2.2 WSDL-S ............................................................. 17
  2.2.3 Web Service Modeling Ontology (WSMO) ..................... 19
2.3 Trust and Policies .................................................... 20
  2.3.1 The Ontology of Trust ........................................... 22
  2.3.2 Policy-based Trust Management ............................... 23
2.4 Policy Languages .................................................... 25
  2.4.1 PeerTrust ......................................................... 25
  2.4.2 PROTUNE ........................................................ 29
  2.4.3 Rei ................................................................. 30
  2.4.4 KAoS (Knowledgeable Agent-Oriented System) .............. 33
  2.4.5 eXtensible Access Control Markup Language (XACML) .... 35
2.5 Web Rule Languages .................................................. 36
  2.5.1 RuleML ............................................................ 40
  2.5.2 Semantic Web Rule Language (SWRL) ......................... 40
2.6 Summary ..................................................................... 42

3 Theoretical Foundations .................................................. 44
  3.1 Rule Interchange Format (RIF) .................................... 45
    3.1.1 The use cases for the RIF .................................... 45
  3.2 REWERSE Rule Markup Language (R2ML) ....................... 49
    3.2.1 Designing The REWERSE Rule Markup Language .......... 49
    3.2.2 R2ML’s Vocabulary ............................................ 51
    3.2.3 R2ML’s Terms .................................................. 52
    3.2.4 R2ML Atoms .................................................... 55
    3.2.5 R2ML Formulas ............................................... 58
    3.2.6 R2ML Actions .................................................. 59
    3.2.7 R2ML Rules ..................................................... 60
  3.3 Theoretical Foundations of Transforming Policies .............. 65
    3.3.1 Comparing KAoS and Rei ..................................... 67
    3.3.2 Description Logic vs. Computational Logic ................ 69
  3.4 Summary ................................................................... 74
4 The Policy Interchange Framework

4.1 Layered Policy Interchange Framework ........................................ 76
4.2 MetaModels for Rei and KAoS .................................................. 79
  4.2.1 Modular Policy Languages .................................................. 80
  4.2.2 Policies for KAoS and Rei .................................................. 81
  4.2.3 Constraints and Conditions in Policy Languages ........................ 84
  4.2.4 Actions in KAoS and Rei .................................................. 86
  4.2.5 Actors in KAoS and Agents in Rei ...................................... 89
  4.2.6 Types of Policies in KAoS and Rei ...................................... 91
  4.2.7 SpeechActs in Rei .......................................................... 94
  4.2.8 The concepts specific to KAoS ........................................... 94
4.3 Transforming the Policies to/from R2ML ..................................... 94
  4.3.1 Transforming Policies from KAoS to R2ML ............................... 97
  4.3.2 Transforming Policies from Rei to R2ML ................................. 106
  4.3.3 Transforming Policies from R2ML to KAoS/Rei .......................... 110
4.4 Summary ................................................................................. 112

5 Examples of Transforming between Policy Rules ................................ 115
  5.1 KAoS Policies to R2ML Derivation Rules .................................... 116
    5.1.1 Printer Access Policy by Sky Team Customers ........................ 116
    5.1.2 Communicating to Service Providers which Support X.509 Certificates 121
  5.2 Rei Policies to R2ML Derivation Rules ...................................... 125
  5.3 R2ML Derivation Rules to Rei Policies ...................................... 127
  5.4 R2ML Derivation Rules to KAoS Policies .................................... 133
  5.5 Summary ................................................................................. 134

6 Analysis of the Policy Interchange Framework .................................. 137
  6.1 Compatibility of our Transformations with other R2ML Transformations ........................................ 138
    6.1.1 R2ML to RuleML ............................................................. 139
    6.1.2 R2ML to Jena ................................................................. 139
    6.1.3 R2ML to F-Logic ............................................................. 144
  6.2 Information Loss during Policy Transformations ............................ 147
  6.3 Rules and Extensions for R2ML to be used as a Policy Rule Language ........................................ 151
  6.4 Summary ................................................................................. 152
7 Conclusions

7.1 Contributions ................................................. 155
7.2 Future Work ................................................. 158

Bibliography ......................................................... 160
List of Tables

2.1 The comparison of PeerTrust, Rei, and KAoS policy languages .............. 36
2.2 Basic Tags in RuleML .............................................. 41

3.1 DL syntax, FOL representation, and the corresponding DAML+OIL constructors ......................................................... 71

4.1 The concrete XML-based conversion of some of the KAoS elements to R2ML elements ......................................................... 105
4.2 The concrete mapping of two of the Rei elements to their corresponding R2ML elements ......................................................... 112

6.1 Transformations between R2ML and other rule languages ................. 138
List of Figures

2.1 The Semantic Layer Cake, the intended structure of the Semantic Web . . . . 11
2.2 The top level of OWL-S service ontology .................................................. 16
2.3 Externalized representation and association of semantics to WSDL elements . 18
2.4 The relation between MOF layers and WSMO ............................................ 19
2.5 The top-level concepts of WSMO ............................................................... 20
2.6 The Ontology of Trust ................................................................................. 22
2.7 The Architecture of PeerTrust [74]. .......................................................... 28
2.8 The Hierarchy of rule types .......................................................... 39

3.1 The notions for the elements of a CFA diagram ............................................. 46
3.2 The goals of the RIF, represented as a CFA diagram ................................. 47
3.3 The CFA diagrams for the RIF’s critical factors ........................................... 48
3.4 The RIF predictability requirement represented as a CFA diagram .................. 48
3.5 Different abstraction levels considered for rules in R2ML ......................... 50
3.6 The representation of R2ML’s vocabulary in UML meta-model ..................... 53
3.7 The UML meta-model for R2ML’s ObjectTerm ......................................... 53
3.8 The UML representation of R2ML’s DataTerm ...................................... 54
3.9 The UML meta-model for R2ML’s GenericTerm ..................................... 55
3.10 The UML meta-model for R2ML variables ............................................ 55
3.11 The UML representation for R2ML Atoms ........................................... 56
3.12 The UML representation of R2ML’s ObjectDescriptionAtom ....................... 56
3.13 The UML meta-model for ObjectClassificationAtom ............................... 56
3.14 AttributionAtom represented as UML meta-model .................................. 57
3.15 ReferencePropertyAtom defined as UML meta-model .............................. 57
3.16 AssociationAtom in R2ML meta-model .................................................. 57
4.22 The meta-model for KAoS Actor .................................................. 91
4.23 The meta-model for KAoS policy types ....................................... 92
4.24 The meta-model for Rei policy types .......................................... 92
4.25 The meta-models for a) an ObligationPolicy and b) an ObligationConstraint in KAoS .............................................................. 93
4.26 The meta-model for KAoS ObligationConstraint types ...................... 93
4.27 The meta-model for a) Rei SpeechAct Types and b) Rei SpeechAct element .............................................................. 94
4.28 The meta-model for KAoS Group Types ........................................ 95
4.29 The meta-model for the Group element in KAoS ............................. 95
4.30 The meta-model for KAoS Domain concept .................................. 95
4.31 The meta-model for KAoS Team concept ..................................... 96
4.32 The meta-models for our R2ML policy classes ............................... 99
4.33 The abstract transformation of a KAoS Policy to a R2ML derivation rule 100
4.34 The abstract model of mapping a KAoS Policy class to our R2ML policy model 100
4.35 The meta-model to describe a R2ML Policy class through ObjectDescription-Atom in R2ML ......................................................... 101
4.36 The abstract transformation of a KAoS property to a R2ML ReferencePropertyAtom .......................................................... 102
4.37 The abstract transformation of a KAoS role-value property to a R2ML ObjectClassificationAtom ............................................ 103
4.38 Using ObjectClassificationAtom and ReferencePropertyAtom to map the variables in KAoS and Rei ........................................ 104
4.39 The abstract model of transforming a Rei policy element to R2ML derivation rule ..................................................... 107
4.40 The abstract meta-model for converting between the element of Rei policy and R2ML policy ................................................ 108
4.41 The abstract mapping model from a Rei SimpleConstraint to R2ML's ReferencePropertyAtom ............................................ 109
4.42 The abstract mapping model from Rei BinaryConstraint to R2ML's logical formula ...................................................... 109
4.43 The abstract mapping of a Rei action to R2ML's ObjectDescriptionAtom .............................................................. 111
4.44 The abstract model of making a KAoS action out of a set of R2ML ReferencePropertyAtoms .................................................. 113
4.45 Abstract model of transforming a R2ML ObjectDescriptionAtom to Rei's SimpleConstraint ................................................................. 114

5.1 The meta-model for KAoS AccessAction ................................................................. 116

6.1 An eCommerce scenario for exchanging policies ................................................. 144
List of Programs

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The main policy element for a Rei Policy Rule</td>
<td>30</td>
</tr>
<tr>
<td>2.2</td>
<td>The constraint elements for a Rei Policy Rule</td>
<td>31</td>
</tr>
<tr>
<td>2.3</td>
<td>A Deontic Element in Rei Representing one single policy rules</td>
<td>32</td>
</tr>
<tr>
<td>2.4</td>
<td>A KAoS Policy as well as the resource that it controls</td>
<td>34</td>
</tr>
<tr>
<td>2.5</td>
<td>A Class definition following KAoS conventions to be used by a KAoS policy</td>
<td>35</td>
</tr>
<tr>
<td>2.6</td>
<td>A Sample Policy defined in XACML (Part1)</td>
<td>37</td>
</tr>
<tr>
<td>2.7</td>
<td>A Sample Policy defined in XACML (Part2)</td>
<td>38</td>
</tr>
<tr>
<td>2.8</td>
<td>A sample rule in RuleML</td>
<td>41</td>
</tr>
<tr>
<td>2.9</td>
<td>Rule Definition in SWRL</td>
<td>42</td>
</tr>
<tr>
<td>3.1</td>
<td>A sample integrity rule in R2ML XML</td>
<td>62</td>
</tr>
<tr>
<td>3.2</td>
<td>A sample derivation rule in R2ML XML</td>
<td>63</td>
</tr>
<tr>
<td>3.3</td>
<td>A sample production rule in R2ML XML</td>
<td>64</td>
</tr>
<tr>
<td>3.4</td>
<td>A sample reaction rule in R2ML XML</td>
<td>66</td>
</tr>
<tr>
<td>4.1</td>
<td>Redefining the ontology classes in Rei</td>
<td>82</td>
</tr>
<tr>
<td>4.2</td>
<td>The R2ML Vocabulary for our defined R2ML Policy Model</td>
<td>98</td>
</tr>
<tr>
<td>5.1</td>
<td>The R2ML ObjectDescriptionAtom excerpt of the KAoS Policy Program 2.4</td>
<td>117</td>
</tr>
<tr>
<td>5.2</td>
<td>The R2ML ClassificationAtom excerpt of the KAoS Policy Program 2.4</td>
<td>119</td>
</tr>
<tr>
<td>5.3</td>
<td>The R2ML ReferencePropertyAtom excerpt of the KAoS Policy Program 2.5</td>
<td>120</td>
</tr>
<tr>
<td>5.4</td>
<td>The R2ML DerivationRule excerpt</td>
<td>120</td>
</tr>
<tr>
<td>5.5</td>
<td>The KAoS policy that restricts the behavior of a broker agent to communicate only to the service providers that support X509 credentials</td>
<td>122</td>
</tr>
<tr>
<td>5.6</td>
<td>The R2ML ObjectDescriptionAtom excerpt of the KAoS Policy Program 5.5</td>
<td>123</td>
</tr>
<tr>
<td>5.7</td>
<td>The R2ML ClassificationAtom excerpt of the KAoS Policy Program 5.5</td>
<td>124</td>
</tr>
<tr>
<td>5.8</td>
<td>The Rei policy that prevents Graduate Students to print in the CS Department</td>
<td>126</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.9</td>
<td>The R2ML ObjectDescriptionAtom for the conclusion part of our derivation rules</td>
<td>127</td>
</tr>
<tr>
<td>5.10</td>
<td>The R2ML ReferencePropertyAtoms for the condition part of our derivation rules</td>
<td>128</td>
</tr>
<tr>
<td>5.11</td>
<td>The Rei equivalent of Program 5.5 through R2ML - Part 1</td>
<td>130</td>
</tr>
<tr>
<td>5.12</td>
<td>The Rei equivalent of Program 5.5 through R2ML - Part 2</td>
<td>131</td>
</tr>
<tr>
<td>5.13</td>
<td>The Rei equivalent of Program 5.5 through R2ML - Concept Matching</td>
<td>132</td>
</tr>
<tr>
<td>5.14</td>
<td>The Rei equivalent of Program 5.8 through R2ML - Part 2</td>
<td>135</td>
</tr>
<tr>
<td>6.1</td>
<td>The RuleML equivalent of Program 5.5 through R2ML and the one-to-one mappings</td>
<td>140</td>
</tr>
<tr>
<td>6.2</td>
<td>The RuleML equivalent of Program 5.8 through R2ML and the one-to-one mappings</td>
<td>141</td>
</tr>
<tr>
<td>6.3</td>
<td>The Jena rule equivalent of Program 5.8 through R2ML and its one-to-one mappings</td>
<td>143</td>
</tr>
<tr>
<td>6.4</td>
<td>The F-Logic rule for the Seller-Buyer scenario</td>
<td>145</td>
</tr>
<tr>
<td>6.5</td>
<td>The KAoS rule for the Seller-Buyer scenario</td>
<td>146</td>
</tr>
<tr>
<td>6.6</td>
<td>The one-to-one mappings between F-Logic and R2ML for the Seller-Buyer scenario</td>
<td>148</td>
</tr>
<tr>
<td>6.7</td>
<td>The one-to-one mapping between KAoS and R2ML for the Seller-Buyer scenario</td>
<td>149</td>
</tr>
<tr>
<td>6.8</td>
<td>The Integrity Rule representation for Program 6.4</td>
<td>153</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background

Semantic Web services (SWS), as the augmentation of Web service descriptions through Semantic Web annotations, facilitate the higher automation of service discovery, composition, invocation, and monitoring on the Web [81]. Semantic Web ontologies and its ontology languages (OWL and RDF(S)) are recognized as the main means of knowledge representation for Semantic Web services [94]. Such ontology-enriched Web service descriptions are later used in the negotiation process between service clients and service providers, which is defined by a set of abstract protocols of Semantic Web Service Architecture (SWSA) [17].

However, the current proposed standards for describing Semantic Web services (i.e. OWL-S [64], WSDL-S [3], Web Service Modeling Ontology [90], and Semantic Web Service Language SWSL [6]) demonstrate that it is important to use a rule language in addition to ontologies. This allows run-time discovery, composition, and orchestration of Semantic Web services by defining preconditions or post-conditions for all Web service messages exchanged [62]. For example, OWL-S recommends using OWL ontologies together with different types of rule languages (SWRL, KIF, or DRS), WSMO uses F-Logic, while WSDL-S is fully agnostic about the use of a vocabulary (e.g., UML, ODM, OWL) or rule language (e.g., OCL, SWRL, RuleML). Usually, Semantic Web service descriptions use only parts of rules representing logical formulas that may have a Boolean result. It is important to point out that there is no agreement upon which rule language to use for Semantic Web services or what type of reasoning (open or closed world assumption) should be supported.
Besides satisfying clients goals when using Semantic Web services, trust is another important aspect that should be established between a client and service. Addressing this problem, researchers proposed the use of policy languages. A policy is a rule that specifies under which conditions a resource (or another policy) might be disclosed to a requester [77]. To define policies on the Semantic Web, various policy languages have been proposed such as PeerTrust [77], KAoS [103], Rei [49], and PROTUNE [13]. As [77] reports, trust management policies are also defined as parts (most commonly preconditions) of Semantic Web service descriptions.

It is obvious that besides various Semantic Web service description languages, we have various policy languages and various rule languages. All these languages are based on different syntactic representations and formalisms with no explicitly defined mapping between them. This hampers the use of Semantic Web services from two different perspectives. One perspective is automatic negotiation between service client agents and service providers and automatic matchmaking, where agents and matchmakers should be able to understand various rule/policy/service web service description languages. Another perspective is that of a knowledge management worker who needs to be able to express the rules and policies in a single form rather than in a broad variety of forms. To attempt to represent the same rules and policies in many forms is cumbersome, time consuming and error prone but it is the only choice currently available if a broad base of interoperability is required.

We start from the presumption that rules encoded in Semantic Web services and access control policies, and business rules are parts of the general set of policy rules [13, 111], and that we should be able to share them by using the same representation. Such a rule language should enable modeling different types of rules such as reaction rules considering the event-driven nature of Web services (or so-called Event-Condition-Action Rules), derivation rules considering the importance of inferring new facts (such as RuleML), and integrity rules considering the deontic, i.e. must, nature of policy languages. Unfortunately, current Web rule markup languages such as RuleML and SWRL are unable to express all these types of rules.
We propose the use of REWERSE Rule Markup Language (R2ML) [114], which addresses almost all use-cases and requirements for a Rule Interchange Format (RIF) [31], along with a set of transformations between Semantic Web service description (e.g., WSDL-S), rule, and trust management policy languages. We illustrate the benefits of our approach using a Semantic Web Service Architecture example where R2ML is used to share Semantic Web service descriptions and policies in the process of matchmaking and trust negotiation. Based on the obtained results, we evaluate and analyze our suggested rule transformation method and compare it with the requirements of RIF.

1.2 Motivations

As we already discussed in Section 1.1, the new generation of policy languages goes beyond defining rules that constrain only the actors of an action rather than the context in which the action is performed. These languages also put constraints on the resources of a domain. The resources may evolve and expand over time, so the policy languages should be expandable as well. Since ontologies are easy to extend, Semantic Web has gained a lot of reputation in this area. Consequently, the intermediary language that is going to be used to transform the information should not only support the definition of the rules but also be able to transfer the domain properties and semantic features. Here, we argue that Semantic Web rules are appropriate solutions to this problem, as they can cover the policies through defining the rules and ontologies. Further on this can be found in [52], where we have discussed and explained the logic of transforming between policy languages.

Most of the proposals on Web rule languages are trying to address the uses cases and requirements defined by Rule Interchange Format Working Group [31].

Rule Interchange Format (RIF) [31] is an initiative by W3C RIF Working Group to address the problem of interoperability between existing rule-based technologies. Besides standardizing the use of an interchange format for rules, RIF is trying to devise an approach which is easily extensible for the future rule technologies and other enabling technologies. RIF is desired to play as an intermediary language between various rule languages and not as a semantic foundation for the purpose of reasoning on the Web. RIF Working Group has defined ten uses cases which have to be covered by a language compliant to the RIF
CHAPTER 1. INTRODUCTION

properties. Three of these use cases are dealing with policies and business rules, namely: Collaborative Policy Development for Dynamic Spectrum Access, Access to Business Rules of Supply Chain Partners, and Managing Inter-Organizational Business Policies and Practices. This shows the important role policies play as a type of rule languages on the web as well as the real need to let systems with different policy languages communicate. SWRL [41] and RuleML [40] are two of the ongoing efforts in this area trying to serve as rule languages to publish and share rule bases on the World Wide Web. We further elaborate on RIF in Section 3.1.

Besides the requirements and use cases defined by RIF working group, enabling different entities (i.e., resources, clients, and broker agents) on the Web to exchange their policies, gives them the opportunity to share their policies in the process of trust negotiation. Trust Negotiation is the process of automatically exchanging the credentials between the peers of negotiation, according to their own declarative, rule-based policies [11]. At each negotiation step, a credential request is formulated essentially by reasoning with the policy, e.g. by inferring implications or computing abductions. However, during the process of negotiation it may occur that either of the peers of negotiation fails in providing the appropriate credentials. Depending on the interest of the other peer in continuing with the negotiation process, it may release some parts of its policies to the failed peer to inform it of the exact set of credentials that are required. In the area of trust negotiation it is referred to as Explanations. Chances are that the language of the released policy is completely different from what is understandable by the peer that receives it, thus there should be a possibility to easily convert the received policies to a language understandable by the target peer. It can be another motivating case for having an intermediary Web rule language to provide transformations between the policies.

In this dissertation, we present REWERSE Rule Markup language (R2ML) as an attempt to address the use cases and requirements for RIF. We show how R2ML can be the appropriate medium for the policy languages to be interchanged and how this interchange can facilitate achieving the goals defined by RIF.
1.3 Scope of Research

Rule Interchange Working Group was established by the World Wide Web Consortium in Summer 2005 and thus is a fairly new working group. As an obvious consequence, the whole efforts on providing exchangeable rules between different entities on the Web, as the main goal of RIF working group, are in their preliminary steps as well. R2ML, as one of the efforts aligned with RIF specifications, is still under development and the missing elements and constructs of the language are constantly under revision.

This project proposes a framework and a proof of concept over the possibility of making the policies interchangeable as a part of the goals to be accomplished by R2ML and RIF. Within the framework, we were aiming to identify the concepts that are being addressed by all policy languages, the supported logics by each policy language, the correspondence between the elements of different policy languages, and finally the problems that might be faced while transforming the policies. We were hoping that these case studies would help with better development of R2ML and making it closer to the standards of RIF working group.

Considering the abovementioned points, the project's goal was not to provide one to one mappings between different elements of the investigated policy languages. It even turned out that such mappings are not possible to obtain unless by extending the policy languages. This was mainly because the level of abstraction in different policy languages was discovered to be different. However, the project can be considered as a break through, due to the non-existence of any effort in the area of interchanging rules between policy languages as well as a proof over the possibility of using R2ML as the medium for carrying the policies. Although our investigation of the policy languages showed that the underlying logic of the policy languages can be very different - e.g. from descriptive logic in one policy language to computational logic in another - we showed that R2ML can be expressive enough to cover the concepts of different policy languages with different logics. At some points also our project showed some possible improvements to R2ML which will appear in the next version of R2ML - R2ML v0.5.
1.4 The Research Question

The underlying research question was to investigate whether it is feasible to interchange policies by using Web rule languages. To the best of our knowledge, there has been no concrete and practical effort in the area of exchanging policy rules between different policy languages. Although there has been some working groups and technical committees formed to investigate the area [82], our effort seems to be the first practical solution to the problem of exchanging the policy rules between different policy languages.

The main goal followed in this research project was to prove that providing the transformations between the policy languages is possible, to show the concepts that will be lost during the process of policy transformation, and to have an analysis over the harms and perils that may occur as a result of this information loss. The research gives us a better understanding of the identical concepts among policy languages and illustrates the points of further development in policy languages in order to obtain a comprehensive semantic model for what policy languages need to cover; to be later used to design a framework which will support high levels of interoperability between policy (and rule) languages.

1.5 The Structure of the Dissertation

This dissertation has been organized in seven chapters. Chapter 1 is to give the reader a brief overview of the project as well as the motivations (Section 1.2), the scope (Section 1.3) and the main research question followed in the project (Section 1.4). Chapter 2 is a review of the related work in the areas that are connected to the area of research in this project. The variety of research areas that have been touched in this project has made this chapter a quite long part of the dissertation. This chapter covers the available research findings in the areas of Semantic Web (Section 2.1) and Semantic Web Services (Section 2.2), the concepts of trust (Section 2.3) and policies along with the most known policy languages (Section 2.4), and also the efforts in the area of Web rule languages (Section 2.5). Chapter 3 explores the formalisms of providing a successful transformation framework between the policy languages. In this chapter we will review the RIF requirements more deeply, describe the formalisms of having a successful and accurate transformation, explain the constructs
CHAPTER 1. INTRODUCTION

of R2ML and argue over how R2ML addresses different concepts in different logics by expanding the discussions over available logics for policy languages.

In Chapters 4 and 5 we define the mapping tables between R2ML constructs and the elements of different policy languages that we have chosen in this project. We also show how the concepts of different policy languages can carry similar meaning during the process of transformation and how one policy rule in a policy language can be converted to a conceptually similar rule in another policy language. We further expand on the transformations by providing some examples of applying the transformations to some real world policy rules described in different policy languages and showing how the meaning of the policy rules will be preserved after applying the transformations. Chapter 6 has been devoted to analyzing the solutions represented in the project. It will also review the proposed solutions, the limitations, the constraints, and the to-be-revised elements of R2ML. This Chapter also deals with different types of R2ML rules that can be used to describe the policy rules and compares them. Finally Chapter 7 is a conclusion to the dissertation. We again review all the explained ideas in the dissertation, summarize the contribution of this research, and define the direction of the future work.
Chapter 2

Policy Management Approaches and Web Service Technologies

Defining transformations between existing policy languages is not just a matter of providing some mapping rules between the elements of different policy languages. Policy languages are considered as a way to accomplish trust which is itself an objective to be addressed by the Semantic Web. Consequently, the technologies and research areas that are partly related to the Semantic Web should be investigated while working on the policies and policy-based systems. This includes the concepts of Semantic Web and Semantic Web languages which will be reviewed in Section 2.1, and also Semantic Web Service as the augmentation of Web services through semantic annotations (Section 2.2). On the other hand, policies can be regarded as an approach to provide security checks and security controls over the access to the resources, the privacy of the users, the quality of services, and etc; thus security is another subject to be explored before dealing with the matters in policy exchange (Section 2.3).

The examination and understanding of available policy languages is of course the most important piece of the puzzle in our research (Section 2.4). At the end one needs to have a clear idea about how different Web rule languages try to cover several concepts and ideas represented in different rule models with regards to their constructs. This will help with having a precise understanding on: 1) Why a Web rule language is appropriate to transform the policy rules, 2) What the purpose in using Web rule languages is, and 3) Whether a Web rule language is comprehensive enough to cover all the concepts we require for the
purpose of completely transforming the rules of different domain specific languages (Section 2.5). This Chapter answers the above questions and gives an overview of the relevant areas of research.

2.1 Semantic Web

The representation of information on the Internet has been shifting from static HTML pages with text only format, to dynamic pages connected to the DataBases, with constant refreshment of their contents through using Java Server Page (JSP), Active Server Pages (ASP), PHP, Perl, and etc. It has been even more evolved by putting multi-media content in the form of videos, audios, images, etc. online, making the Internet a rich collection of all types of data and information. However, the view of the Internet should be separated from the stored content, for machines to be able to interpret the information.

The term Semantic Web was first coined by Tim Berner-Lee in [7]. The main idea behind creating a Semantic Web network is to change the currently unstructured data available on the World Wide Web to a structured and machine understandable model so that intelligent agents can easily traverse the Web and extract the required data. To create such a network, knowledge should be organized in a meaningful and machine understandable structure, referred to as ontology.

Ontology is the formal and explicit specification of conceptualization of a domain [37] into human understandable but machine readable format. However a more recent definition states that an ontology is a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic [39]. It is usually defined as the quadruple of entities, attributes, relationships, and axioms [84]. We further elaborate on the meaning of ontologies in Section 2.1.1

Passin in his book [80] counts the characteristics below for the Semantic Web by referring to some other researchers’ work:

- *The machine-readable-data view:* The idea of having data on the Web has been defined and linked in a way that it can be used by machines not just for display purposes, but also for automation, integration, and reuse of data across various applications [109].
• **The intelligent agents view**: Semantic Web aims at providing annotations for the content of the Web in order to enable the software agents to dynamically, automatically, and intelligently retrieve the information [21].

• **The distributed database view**: The Semantic Web concept is to do for data what HTML did for textual information systems: to provide sufficient flexibility to be able to represent all databases and logic rules to link them together to great added value [108].

• **The automated infrastructure view**: Semantic Web is an infrastructure and not an application [8].

• **The servant-of-humanity view**: The Semantic Web is considered as a vision to relieve the human being from much of a burden in searching, locating, using, and storing the Web content [22].

• **The better-annotation view**: The idea of a “Semantic Web” supplies the (informal) Web with annotations expressed in a machine-processable form and linked together [28].

• **The improved searching view**: Soon it will be possible to access the Web resources by content rather than just by keywords [5].

• **The Web services view**: Increasingly, the Semantic Web will be called upon to provide access not just to static documents that collect useful information, but also to services that provide useful behavior [56]. The Semantic Web promises to expand the services for the existing Web by enabling software agents to automate procedures currently performed manually and by introducing new applications that are infeasible today [28].

According to the abovementioned points, Semantic Web covers a wide range of areas with many different opinions about it. Nevertheless, referring to the descriptions above, one can recognize the following goals in extending and developing Semantic Web [80]: 1) indexing and retrieving information, 2) metadata, 3) annotation, 4) the Web as a large, interoperable database, 5) machine retrieval of data, 6) Web-based services, 7) discovery of services, and 8) intelligent software agents. Consequently, the structure of the Semantic Web can be visually depicted as in 2.1, (aka the Semantic Layer Cake).
2.1.1 Semantic Web Ontologies

The ontologies, as introduced in Section 2.1, are to represent the knowledge. The main purpose in using ontologies is to enable the reuse and sharing of the knowledge that is already stored and structured, instead of reworking all the initial steps [30]. The development of ontologies is pretty much the same as the traditional approaches of developing software systems [75]. Some examples of such systems are object-oriented systems and databases. In essence, development of ontology consists of a domain analysis. Although software and engineering processes have different aspects, convergence between those disciplines is getting more and more apparent. As a result, the latest research suggests using software engineering techniques for ontology development (e.g. UML [58], and software patterns [26]).

Specification of the ontology in the knowledge systems have two dimensions: domain-factual knowledge and knowledge for solving problems [19]. Kalfoglou ranks the ontologies according to their purpose [50]. In the first group there are ontologies for knowledge presentation, and an example of such an ontology is the Frame ontology [37], which specifies the primitives used in frame-based languages. In the second group are task ontologies which specify the knowledge of a task that is independent from the domain in which the task can be performed [71]. Method ontologies complement task ontologies as they provide definitions of relevant concepts and relations that are used for defining reasoning processes. In the early research of ontologies, list-like syntax has been used for representing ontology knowledge. An example of such a language is KIF (Knowledge Interchange Format). However, the
appearance of eXtensible Markup Language (XML) as a specification for data interchange and interoperability on the Web, affected ontology languages in a way that those languages are now using XML-based syntax.

2.1.2 Semantic Web Ontology Languages

Semantic Web was first established based on the standard language for data interchange, i.e. eXtensible Markup Language (XML). The basic idea in XML is to define a standard that will be used for interchanging the content and not the view of the data on the Web. XML, as a metalanguage (a language to define the other languages), has no predefined keywords, i.e. no predefined elements and attributes. Furthermore, XML to be efficiently used, has employed a series of other languages and technologies as well [43], which include:

- **XPointer**: enables addressing a certain part in an XML document;
- **XLink**: a standard for connecting XML documents;
- **XPath**: a standard that uses XPointer to specify paths to locations being addressed;
- **namespace**: a standard used to avoid name collisions;
- **eXtensible Stylesheet Language (XSL)**: a standard that exists for XSL Transformations (transforming one XML document into another), and a language for formatting, XSL Formatting Objects (XSL FO).

In order to be correct, an XML model has to be well formed and valid. The first criterion implies compliance to general syntax rules of XML, such as enclosing attribute values in quotation marks, or ensuring that all elements are balanced (each opened element must be closed). The second criterion means that the document must be written according to some predefined specific grammar. To this end, two W3C standards are used: Document Type Definition (DTD) and XML Schema. DTD specifies how the elements of an XML document are defined, what the attributes of an element are, and what their contents are. DTD has many limitations [91]. For example, it does not allow to define the number of elements that can be contained in another element. Likewise, there is no support for datatypes. Because of such limitations, the XML Schema standard was created. It overcomes DTD's problems, defines additional functionalities, 44 datatypes, and support for inheritance and precise definition of multiplicity. However, this is still not enough for representing ontologies, because
semantic units for a specific domain can not be recognized. In addition, it is very difficult to separate semantic relations between different concepts which exist in a domain, and it is difficult to create mappings between two domains, because both are defined with XML [25, 57]. To solve these problems, W3C consortium recommended RDF and RDFS languages.

Resource Description Framework (RDF) is a W3C standard [61] created to standardize defining and using metadata, i.e. resource descriptions. RDF provides data model that supports a fast integration between data sources, thereby bridging semantic differences. As its name suggests, RDF is not a language but a model for representing data about things on the Web [55]. All elements that RDF describes are called resources. A resource can be anything that a URI can denote as a resource. The basic building block in RDF is the object-value-attribute (O-A-V) triple, which is often written as $A(O, V)$. This means that some object $O$ has attribute $A$ with value $V$.

RDF Schema (RDFS) [16] can be used to define vocabulary of RDF documents, and thus specify object types to which a certain property can be applied. This means that RDFS provides a basic typing mechanism for RDF models. To enable reasoning services for the Semantic Web, another layer above RDF(S) is needed. This (logical) layer introduces ontological languages (see 2.1), which are based on meta-modeling of the adjacent lower layer. It introduces a richer set of modeling primitives, which can be mapped to description logics. This enables using tools with generic support for reasoning that is independent of the problem domain. Examples of early languages of this kind are OIL and DAML [42]. More recently, Web Ontology Language (OWL) is adopted by W3C as a standard ontology representation language for the Semantic Web.

Web Ontology Language (OWL) is a semantic language for publication and sharing of ontologies on the World Wide Web. OWL is developed by extending the RDF vocabulary and based on the experiments with DAML+OIL Web ontology language [24]. Since WWW is unlimited, OWL must start from the open-world assumption and enable inclusion and inference of different ontologies. Some of them might be contradictory, but new information must not overrun the existing one, and may be only added. To enable such possibilities, and simultaneously support computing and reasoning in a finite time with tools that can be
CHAPTER 2. POLICY MANAGEMENT AND WEB SERVICE TECHNOLOGIES

built with existing or forthcoming technologies, OWL introduces three sublanguages with different expressivity for different purposes:

- **OWL Lite** is OWL DL (cf. next bullet point) with more restrictions. The idea is to make it easy to start with, and easy to implement processors for, so that people can begin using OWL Lite easily and later migrate to more complex usage.

- **OWL DL (Description Logic)** enables maximum expressivity and simultaneously ensures completeness of computation and decidability. Completeness means that all connections can be solved in finite time. A limitation to this sublanguage is that, unlike OWL Full (cf next bullet point), OWL DL does not allow a class to be an individual and a property at the same time.

- **OWL Full** provides maximum expressivity and syntax divergence from RDF. The main characteristic of OWL Full, in contrast to OWL DL and OWL Lite, is that class, which is defined as a collection of individuals, can be an individual itself, as in RDF(S).

2.2 Semantic Web Services

As mentioned in Section 2.1, the original motivations of the Semantic Web were to increase automation in processing Web-based information and to improve interoperability of Web-based information systems. Web services are undeniably of the most important Web resources that should be annotated. The term “Web service” refers to the Web sites or Web resources that go beyond providing only static information, and enable the users to make a change in the world, e.g. to control a physical device or to perform some actions. Service Oriented Architecture (SOA) provokes dynamic discovery and invocation of those Web services that allow to solve particular requests [27]. Three key technologies that make the goals of SOA attainable are *Web Service Description Language (WSDL)* [20], *Simple Object Access Protocol (SOAP)* [38], and *Universal Description Discovery and Integration (UDDI)* [100]. WSDL is a description for the service which defines, in- and outgoing messages, the functions, the type of the variables, etc. required by the service to function. SOAP is an XML data exchange protocol for the Web to transfer the information between various services an client requesters. Finally UDDI is a registry or repository for the Web services to be looked for.
Semantic Web services (SWS), as the augmentation of Web service descriptions through Semantic Web annotations, facilitate the higher automation of service discovery, composition, invocation (execution), and monitoring on the Web [81]. Automatic Service Discovery is generally accomplished by automatically locating the Web services that provide a particular service and adhere to the requested properties. Automatic Service Composition is defined as automatic selection, composition, and interoperation of appropriate Web services to perform some task, given a high-level description of the task's objective. Automatic Service Execution happens when a computer program or agent automatically executes an identified Web service. And finally, Automatic Service Monitoring can be obtained through automatically checking the behavior of a service over time, and recognizing the deficiencies, flaws, and adjusting the possible harms and problems [65].

For Semantic Web service technology to become true, the content of a Web service, the users of this Web service and the groups they belong to, and also the procedures that are used by automatic broker agents to access the service, must be semantically marked and annotated. Web sites should be able to employ a standard ontology, consisting of a set of basic classes and properties, for declaring and describing services. Annotating the content, the group of users, and the defined procedures of a Web service with regards to this ontology, gives the broker agent a better understanding of what the service is, what it is used for, and what the inputs, outputs, and the effects of executing the service to the real world are. This gives a general understanding of the behavior of the Web service to the broker agent which later can be used to achieve the goals defined for the Semantic Web services, i.e automatic discovery, composition, orchestration, and execution. Next the languages used to annotate the Web services will be reviewed.
2.2.1 OWL-S

OWL-S\(^1\) has been built upon the belief that the ontology structuring mechanisms of OWL provide an appropriate, Web-compatible representation language framework within which it is possible to accomplish the goals of Semantic Web service [64]. The definition of the OWL-S ontology has been motivated by the need to provide three essential types of knowledge about a service.

- What does the service provide for prospective clients?
- How is it used?
- How does one interact with it?

For these questions to be answered, OWL-S defines three service model ontologies, namely, Service Profile, Service Modeling, and Service Grounding (see Figure 2.2).

![Figure 2.2: The top level of OWL-S service ontology](image)

Service Profile provides the information needed for an agent to discover a service, while the Service Model and Service Grounding, taken together, provide enough information for an agent to make use of a service, once found.

\(^1\)The content of this section, including the figures, has been taken from http://www.w3.org/Submission/OWL-S/. Copyright © 2004 France Telecom, Maryland Information and Network Dynamics Lab at the University of Maryland, National Institute of Standards and Technology (NIST), Network Inference, Nokia, SRI International, Stanford University, Toshiba Corporation, and University of Southampton. All Rights Reserved. The document is available under the W3C Document License which grants permission to copy, and distribute the contents of this document in any medium for any purpose and without fee or royalty.
2.2.2 WSDL-S

WSDL-S\(^2\) is another effort in adding semantics to the definition and description of Web services \([3]\). The developers of WSDL-S argue the following points as the problems with OWL-S and consider WSDL-S as an approach to overcome these problems: "First, the OWL-S profile model duplicates the descriptions embodied in the rest of WSDL (namely input and outputs). This leads to the inconvenience of creating multiple definitions for describing the same service. Second, it assumes that everyone uses OWL for representing ontologies which may not always be the case."

WSDL-S is an augmentation of the expressivity of WSDL with semantics by employing concepts analogous to those in OWL-S while being agnostic to the semantic representation language. The advantage of this approach to adding semantics to WSDL is multi-fold. First, users can, in an upwardly compatible way, describe both the semantics and operation level details in WSDL - a language that the developer community is familiar with. Second, by externalizing the semantic domain models, we take an agnostic approach to ontology representation languages. This allows Web service developers to annotate their Web services with their choice of modeling language (such as UML or OWL). This is significant since the ability to reuse existing domain models expressed in modeling languages like UML can greatly alleviate the need to separately model semantics. Moreover, this approach realizes the need for the existence of multiple ontologies, either from the same or different domains to annotate a single Web service and provides a mechanism to do so.

WSDL-S provides a mechanism to annotate the capabilities and requirements of Web services with semantic concepts referenced from a semantic model. To do this, it provides mechanisms to annotate the service and its inputs, outputs and operations. Additionally, it provides mechanisms to specify and annotate preconditions and effects of Web Services. These preconditions and effects together with the semantic annotations of inputs and outputs can enable automation of the process of service discovery. Figure 2.3 shows how

---

\(^2\)The content of this section, including the figures, has been taken from http://www.w3.org/Submission/WSDL-S/. Copyright © 2005 University of Georgia Research Foundation, Inc., International Business Machines Corporation. All rights reserved. The document is available under the W3C Document License which grants permission to copy, and distribute the contents of this document in any medium for any purpose and without fee or royalty.
semantic annotations are associated with various elements of a WSDL document (including inputs, outputs and functional aspects like operations, preconditions and effects) by referencing the semantic concepts in an external domain semantic model. The domain model can consist of one or more ontologies.

Figure 2.3: Externalized representation and association of semantics to WSDL elements

WSDL-S provides five main elements to extend WSDL:

- *modelReference* is an extension attribute to specify the association between a WSDL entity and a concept in some semantic model. It can be added to a complex type, element, operation, as well as the extension elements - *precondition* and *effect* (see next).

- *schemaMapping* is another extension attribute which is added to XSD elements and complex types, for handling structural differences between the schema elements of a Web service and their corresponding semantic model concepts.

- *precondition* and *effect* are primarily used in service discovery, and are not necessarily required to invoke a given service.

- *category* is also another extension attribute which consists of service categorization information that could be used when publishing a service in a Web Services registry.
such as Universal Description Discovery and Integration (UDDI) [100]. Semantic
categorization of UDDI registries using ontologies was proposed in [95, 105].

2.2.3 Web Service Modeling Ontology (WSMO)

WSMO \(^3\) is a comprehensive framework for Semantically Empowered Service-Oriented Ar­
chitecture [90]. WSMO provides ontological specifications for the core elements of Semantic
Web services as well as a meta-model for Semantic Web services related aspects. WSMO has
been specified using *Meta Object Facility Specification (MOF)* [72] which is an abstract lan­
guage and framework for specifying, constructing, and managing technology neutral meta-
models.

In terms of the four layers of MOF pyramid, the language defining WSMO corresponds
to the meta-meta model layer, WSMO itself constitutes the meta-model layer, the actual
ontologies, Web services, goals, and mediators specifications constitute the model layer, and
the actual data described by the ontologies and exchanged between Web services constitute
the information layer. Figure 2.4 shows the relation between different layers of MOF and
WSMO.

![Figure 2.4: The relation between MOF layers and WSMO](image)

WSMO identifies four top level elements as the main concepts which have to be described
in order to describe Semantic Web services 2.5.

---

\(^3\)The content of this section, including the figures, has been taken from
https://www.w3.org/Submission/WSMO/. Copyright © 2005 DERI Innsbruck at the Leopold-Franzens-
Universität Innsbruck, Austria, DERI Galway at the National University of Ireland, Galway, Ireland, BT, The
Open University, and SAP AG. All rights reserved. The document is available under the W3C Document
License which grants permission to copy, and distribute the contents of this document in any medium for
any purpose and without fee or royalty.
Ontologies provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of discourse. Web services describes the computational entity providing access to services that provide some value in a domain. These descriptions comprise the capabilities, interfaces and internal working of the Web service. All these aspects of a Web Service are described using the terminology defined by the ontologies. Goals represent user desires, for which fulfillment could be sought by executing a Web service. Ontologies can be used for the domain terminology to describe the relevant aspects. Goals model the user view in the Web service usage process and are therefore a separate top-level entity in WSMO. Finally, Mediators describe elements that overcome interoperability problems between different WSMO elements. Mediators are the core concepts to resolve incompatibilities on the data, process and protocol level, i.e. in order to resolve mismatches between different used terminologies (data level), in how to communicate between Web services (protocol level) and on the level of combining Web services (and goals) (process level).

2.3 Trust and Policies

As we stated earlier, policies used to establish a certain level of trust between the entities involved in the process of communication. To be able to understand the underlying usage of policies, we first need to know how trust is related to policies and what is the role of the policies in establishing trust between the interacting entities.
According to [1], “Trust is the reliance on the integrity, ability, or character of a person or a thing in a social and physical system”. It can also be expressed as “the firm belief in the reliability or truth or strength of an entity” [74].

Virtual Organizations (VOs) are as real as communities that meet physically or whose members exist in near convenient proximity. Thus, whatever role trust plays in these physical communities also applies to VOs as ultimately all virtual interactions are human bound [2]. Grandison [34] gives a technical definition of trust in VOs as “the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context (assuming dependability covers reliability and timeliness)”. In a VO, trusting a second party is often described as a relationship between a trustee, i.e. the entity that is trusted, and a trustor, the subject that trusts the target entity, on a service or resource of information. Level of trust in a trustee by the trustor depends on the context of interaction.

Generally, in cyber space several characteristics are identified for trust. Below a list of the most important characteristics of trust have been enlisted:

- **Bilateral and Asymmetric**: For entities to start the process of communication, they should have a certain level of trust in each other (Bilateral), but this level of trust is not necessarily the same (Asymmetric).

- **Intransitive**: If party A trusts party B and party B trusts party C, it doesn’t mean that party A will trust party C.

- **Dynamic**: The level of trust between two parties constantly changes over time as a result of their interactions.

- **Continuous or Discrete**: Although trust in the real life is continuous but in the cyber space it is easier to make it a discrete concept, that is, by defining different groups with different levels of trust in each group.

The notion of Risk is in an inverse relation with the notion of Trust. The Risk is the level of jeopardizing the privacy and security for the sake of dealing with a particular internal or external situation. An internal situation could be anything relevant to the relation between the trustor and the trustee, such as gaining more trust, while an external situation
is beyond the relation between the two parties; e.g. it could be releasing information in a
disastrous situation. Entities can choose to trade their privacy for a corresponding gain in
their partners’ trust. Of course the scope of entity’s privacy disclosure should be propor­
tional to the benefits expected from the interaction. Risk aversive policies may set severe
constraints over the revealed information while in a risk taking policy the constraints are
more relax.

2.3.1 The Ontology of Trust

Trust is evaluated based on the knowledge of the trustor about the context of the received
information, the content of the received information, and also the source of information
[97]. Context of information contains meta-information about circumstances such as date,
time, and location. It also appears in some literature that the information about the source,
i.e. the provenance as well as the relation of the provenance with the trusting agent, is
considered as part of the context of information. Considering the concepts presented above,
one can define trust as a predicate over knowledge (K), provenance (P), context (C), and
content (C), of information T(K, P, C, C) [2].

The abovementioned predicate then can be used to define the ontology of trust, gener­
ally used for determining important parameters that influence the evaluation of trust. Grn
[32] has introduced trust as an ontology of Trustor, Trustee, Trust Value, Start Date, and
End Date (Figure 2.6). Dillon, Chang, and Farookh in [97] define a more detailed ontol­
ogy. In their ontology they relate trust to Trusting Peer, Trusted Peer, Context, Time Slot
Start Time, Time Slot End Time, Association Type, and Trustworthiness Value. In their
model, Association Type relationship distinguishes among whether trust has been established between two individual entities, an individual and a group of individuals, two groups of individuals, or only within a peer. They have considered trustworthiness as a dynamic concept that should be kept up to date.

Having all the above properties, attributes, and definitions, we are still facing the question of how to use all this information to provide a more reliable system with better detection of fraudulent behavior and guarantee the competence of the trustees. Several efforts in this area are addressed under the name of Trust Management.

2.3.2 Policy-based Trust Management

The term "distributed trust management" was first coined by [9] as a unified approach to specifying and interpreting security policies, credentials, and relationships which allow direct authorization of security-critical actions. Later in 2003 Grandison in [33] extended the idea and defined it as The activity of collecting, encoding, analyzing, and presenting evidence relating to competence, honesty, security, or dependability with the purpose of making assessments and decisions regarding trust relationships. The concept of trust management has been also referred to as Trust Act. For example [4] addresses the Trust Act as to decide on whether or not to trust.

Trust management formally discriminates strong security mechanism from soft computational approaches. Two different trust management approaches have been developed based on the context of different environments, either with a leaning more toward strict policies of access control and authentication, or with a tendency to soft policies of authorization [87].

Hard policy approaches use objective strong security mechanisms such as signed certificates and trusted certification authorities in order to regulate the access of users to services. The access decisions in this model are usually based on mechanisms with well defined semantics (such as logic programming) and the results are mostly binary decisions. Systems using Public Key Certificates, e.g. methods that use X.509 schema, Role Based Access Control [92], and firewalls are examples of hard policy methods.

On the other hand, reputation-based methods rely on soft computational approaches to the problem of trust. In this case trust is computed from direct interactions with the trustee,
recommendation of other parties in the network, or a combination of both. This method, according to the literature addressing this issue, is mostly used in provisional trust where access to a service or a resource is requested by a party. Considering the requesting party as a client and the data provider as a server, this method is mostly used when a request from a client to the server happens. Peer-2-Peer networks, Virtual Pervasive Communities, and mobile ad-hoc networks are samples of systems using reputation-based methods. In a hard security policy the authorization certificate is signed by a third party which is trusted by both sides, but in a reputation based model the request is self signed and the requesting party holds the responsibility of interaction. This property assists with computation of reputation because in case of a failure or success there is no other party involved and only the reputation of the trusted party will be manipulated.

Hard policy approaches are being used in the structured environments, where there is a limited, most likely known, number of individuals in the environment, with the roles of groups, communities, and individuals precisely determined. Alternatively, reputation-based methods are used in unstructured domains with unknown number of individuals interacting together. The unpredictable number of parties involved in the environment makes it hard, or even impossible, to predefine specific roles. These systems rely on the feedback received from the community and also on the a priori knowledge acquired as a result of direct experience. The results of the new interaction, combined by the a priori information, makes a posteriori evaluation factor for further data exchange. It is obvious in the reputation-based systems that the number of individuals in a community is in a direct relation with the better evaluation of trust factors. These methods rely on the availability of a pool of users interacting together. The results of these interactions are recorded to make the a priori knowledge for other entities.

The reputation-based methods are deemed to be a solution to the currently rigorous constraints with the lack of flexibility in access management. The implicit inclusion of trust management and belief values to sent queries is the goal of trust management community. A synthesis of the two approaches [12], although yet has not been widely adapted, can increase the control on data disclosure.

Although the areas of reputation based and policy based trust management are broad
and there is a lot of ongoing research in either of these two areas, in this dissertation we narrow our focus down to the policy-based trust management in general, and policy languages in particular. The policy languages are the major technology for the purpose of security and privacy control at the moment. The process of information exchange between the existing policy languages, as it was motivated in Section 1.2, can make the distributed policy development, trust negotiation, and failure explanation happen. The rest of this document puts more emphasis on the taken approaches and methods.

2.4 Policy Languages

In this section we briefly review the most known policy languages developed and being used for managing trust in distributed environments.

2.4.1 PeerTrust

PeerTrust [74, 77] is a trust negotiation engine for semantic web and P2P networks. PeerTrust’s language is based on first order Horn rules (definite Horn clauses), i.e. rules of the form

\[ \text{lit}_0 \leftarrow \text{lit}_1, ..., \text{lit}_n \]

where each \( \text{lit}_i \) is a positive literal of the form \( P_j(t_1, ..., t_n) \), \( P_j \) is a predicate symbol, and \( t_i (i = 1..n) \) are the arguments of this predicate. Each \( t_i \) is a term, i.e. a function symbol and its arguments, which are themselves terms. The head of a rule is \( \text{lit}_0 \), and its body is the set of \( \text{lit}_i (i = 1..n) \). The body of a rule can be empty.

Definite Horn clauses are the basis for logic programming and can be easily extended to include negation as failure, restricted versions of classical negation, and additional constraint handling capabilities such as those used in constraint logic programming. Although all of these features can be useful in trust negotiation, the introduced features instead focus on more unusual language extensions of PeerTrust [14].

**Reference to Other Peers.** The ability to reason about statements made by other peers is central to trust negotiation. To express delegation of evaluation to another peer, each literal \( \text{lit}_i \) is extended with an additional *Authority* argument
where Authority specifies the peer who is responsible for evaluating $lit_i$ or has the authority to evaluate $lit_i$. So the literal below specifies that “SFU” is responsible to evaluate “Nima Kaviani” as a student:

$$student(“Nima Kaviani”) @ “SFU”$$

The Authority argument can be a nested term containing a sequence of authorities which are then evaluated starting at the outermost layer (i.e. from the left). For example in the literal below:

$$type(CredentialIdentifier, “SFU Student ID”) @ Issuer @ Requester$$

it is shown that the Requester is the authority responsible for checking the equality of CredentialIdentifier (which is authorized by the Issuer) with “SFU Student ID”. A specific peer may need a way of referring to the peer who asked a particular query. It is accomplished with Context literals that represent release policies for literals and rules, so that now literals and rules are of the form:
CHAPTER 2. POLICY MANAGEMENT AND WEB SERVICE TECHNOLOGIES

\[ \text{lit}_i \oplus \text{Authority} \oplus \text{context}_j \]
\[ \text{lit}_i \leftarrow \text{context}_j \text{ lit}_{i-1}, \ldots, \text{lit}_1 \]

For example, suppose that "Nima Kaviani" has derived a clause \( C \), say his student ID, and he wishes to send this literal to "SFU-ELearning System". He can only do so if he is able to derive \( \text{StudentID} \oplus \text{Requester} = \text{SFU-ELearningSystem} \). That is to say, it is supported only by the requestor, i.e. "SFU-ELearning System", and the trustor, i.e. "Nima Kaviani" while they are negotiating in the same context. Here, Requester is a pseudo-variable whose value is automatically set to the party that "Peer1", i.e. "Nima Kaviani" in our example, is trying to send the literal or rule. If no context is specified for a literal or a rule, the default context \( \text{Requester} = \text{Self} \) applies, implying that the literal or rule cannot be sent to any other peer. "Self" is a pseudo-variable whose value is a distinguished name of the local peer. The release policy for a literal can be cleanly specified in rules that are separate from those used to derive the literal, e.g.,

\[ p(X_1, \ldots, X_n) \oplus \text{context}_p (X_1, \ldots, X_n, \text{Requester}, \text{Self}) \leftarrow p(X_1, \ldots, X_n) \]

In the above example, \( X_i(i = 1..n) \) represents the available resources that are released to the requestor in the context of negotiation. These resources are specified as literals in predicate \( p \).

Using the Authority and Context arguments, evaluation of literals can be delegated to other peers and also interactions and the corresponding negotiation can be easily expressed.

**Signed Rules.** Each peer defines a policy for each of its resources, in the form of a set of definite Horn clause rules. These and any other rules that the peer defines on its own are its local rules. A peer may also have copies of rules defined by other peers, and it may use these rules in its proofs in certain situations.

A signed rule has an additional argument that says who signed the rule. The cryptographic signature itself is not included in the logic program, because signatures are very large and are not needed by this part of the negotiation software. The signature is used to verify that the issuer really did issue the rule. We assume that when a peer receives a signed rule from another peer, the signature is verified before the rule is passed to the DLP.
evaluation engine. Similarly, when one peer sends a signed rule to another peer, the actual signed rule must be sent, and not just the logical programmatic representation of the signed rule. More complex signed rules often represent delegations of authority.

Implementation. PeerTrust 1.0’s outer layer is a signed Java application or applet program, which keeps queues of propositions that are in the process of being proved, parses incoming queries, translates them to the PeerTrust language, and passes them to the inner layer. Its inner layer answers queries by reasoning about PeerTrust policy rules and certificates using Prolog meta-interpreters (in MINERVA and XSB Prolog, whose Java implementation offers excellent portability), and returns the answers to the outer layer. PeerTrust 1.0 imports RDF metadata to represent policies for access to resources, and uses X.509 certificates and the Java Cryptography Architecture for signatures. It employs secure socket connections between negotiating parties, and its facilities for communication and access to security related libraries are in Java. Figure 2.7 shows current PeerTrust architecture.

![Figure 2.7: The Architecture of PeerTrust [74].](image)

As mentioned above RDF metadata is used in PeerTrust as an instantiation of trust ontology, but the system still uses rule-based decision making algorithms as the negotiation language based on Horn clauses shown above.
2.4.2 PROTUNE

PROTUNE or PROvisional TrUst Negotiation is a successor of PeerTrust with the purpose of integrating automated trust negotiation and business rules [13]. It has also been developed on top of PAPL which was another powerful policy language for trust negotiation in 2002. The main contributions with defining PROTUNE are described by the authors as follows:

- Defining a management language supporting general provisional-style actions
- An extensible declarative meta-language for driving decisions about request formulation, information disclosure, and distributed credential collection
- A parametrized negotiation procedure that gives a semantics to the meta-language and provably satisfies some desirable properties for all possible meta-policies
- Integrity constraints for negotiation monitoring and disclosure control
- Ontology-based techniques for importing and exporting meta-policies and for smoothly integrating language extensions.

PROTUNE combines meta-policies and meta-rules using a list of predicates including Decision Predicates, Abbreviation/Abstraction predicates, Constraint predicates, State predicates, and Provisional predicates. Combination of these predicates enables the policy designer to define more fine-grained policies. External resources (such as Databases and other data sources) can be also dynamically queried in this language. Below an example rule of a policy written in PROTUNE is shown.

\[
\text{allow}(Srv) \leftarrow \text{session}(ID), \text{in}(X, \text{sql : query(select * from low - selling)}), \text{enabled}(\text{discount}(X), ID)
\]

The above rule is a decision rule, allow, which grants the access to the peer Srv, a state query, in, which checks the current situation by dynamically querying the information stored in the database, and two provisional queries, session and enabled, the first one checks the ID of the current trust session by getting the information from negotiation context and the second one transfers the current state to a discount state for all Xs that satisfy the query to the database.
2.4.3 Rei

Rei is a policy framework that permits to specify, analyze, and reason about declarative policies defined as norms of behavior [46, 47, 48]. Rei adopts a rule-based approach to specify semantic policies. Rei policies restrict domain actions that an entity can/must perform on resources in the environment, allowing policies to be developed as contextually constrained deontic concepts, i.e., right, prohibition, obligation and dispensation. The first version of Rei (Rei 1.0) was defined entirely in first order logic with logical specifications for introducing domain knowledge. The current version of Rei (Rei 2.0) adopts OWL-Lite to specify policies and can reason over any domain knowledge expressed in either RDF or OWL [88].

A policy basically consists of a list of rules and a context that is used to define the policy domain. Rules are expressed as OWL properties of the policy. In particular, the policy:grants property is used to associate a deontic object with a policy either directly or via a policy:Granting element. Program excerpt 2.1 shows the Rei 2.0 policy specification of the FileAccess policy. In order to specify context conditions, one or more constraints must be defined. A constraint, which may be simple or boolean, i.e., the boolean combination of a pair of simple constraints, defines a set of actors or a set of actions that fulfill a certain property. A simple constraint, as shown in the snippet of Program 2.2, is modeled as a triple consisting of a subject, a predicate and an object, which defines the value of the property for the entity, following a pattern that is typical of logic programming languages like Prolog.

```
<policy:Policy rdf:ID="FileAccessPolicy">
  <policy:actor rdf:resource="#requester"/>
  <policy:grants rdf:resource="#Perm_FileAccess"/>
</policy:Policy>

<policy:Policy rdf:ID="FileSharingPolicy">
  ...
</policy:Policy>
```

Program 2.1: The main policy element for a Rei Policy Rule

A constraint can be associated to a policy at three different levels. The first possibility is to impose a constraint within the definition of a deontic object, by means of the deontic:constraint property, as shown in Program 2.3. In this case, the constraint can be
expressed over the actor, the action to be controlled, or over generic environmental states, e.g., the time of the day. Additional constraints can be imposed within the Granting specification over the entity the granting is made to, the deontic object the granting is made over, and again, over generic environmental states. Finally, it is possible to express a set of constraints directly within the policy definition through the policy:context property. These constraints are generically defined as conditions over attributes of entities in the policy domain.

```xml
<entity:Variable rdf:ID="requester">

<constraint:SimpleConstraint rdf:ID="LocationOfUser">
    <constraint:subject rdf:resource="&some-ontology;user"/>
    <constraint:predicate rdf:resource="&some-ontology;location"/>
    <constraint:object rdf:resource="#user-location"/>
</constraint:SimpleConstraint>

<constraint:SimpleConstraint rdf:ID="CoLocatedWithUser">
    <constraint:subject rdf:resource="#requester"/>
    <constraint:predicate rdf:resource="&some-ontology;location"/>
    <constraint:object rdf:resource="#user-location"/>
</constraint:SimpleConstraint>

<constraint:And rdf:ID="Constraint_CoLocated">
    <constraint:first rdf:resource="#LocationOfUser"/>
    <constraint:second rdf:resource="#CoLocatedWithUser"/>
</constraint:And>
```

Program 2.2: The constraint elements for a Rei Policy Rule

Rei 2.0 uses OWL-Lite for the specification of policies and of domain-specific knowledge. Though represented in OWL-Lite, Rei still allows the definition of variables that are used as placeholders as in Prolog. In fact, as shown in Program 2.2, the definition of constraints follows the typical pattern of rule-based programming languages, like Prolog, i.e., defining a variable and the required value of that variable for the constraint to be satisfied. In this way, Rei overcomes one of the major limitations of the OWL language, and more generally of description logics, i.e., the inability to define variables. For example, as shown in Program 2.3, Rei allows developers to express a policy stating that a user is allowed to access the shared files of another user if they are located in the same area, whereas pure OWL
would not allow for the definition of the "same as" concept. Therefore, Rei's rule-based approach enables the definition of policies that refer to a dynamically determined values, thus providing greater expressiveness and flexibility to policy specification.

```xml
<deontic:Permission rdf:ID="Perm_FileAccess">
  <deontic:actor rdf:resource="#requester"/>
  <deontic:action rdf:resource="&some-ontology;AccessToSharedFiles"/>
  <deontic:constraint rdf:resource="#Constraint_CoLocated"/>
</deontic:Permission>
```

Program 2.3: A Deontic Element in Rei Representing one single policy rules

On the other hand, the choice of expressing Rei rules similarly to declarative logic programs prevents it from exploiting the full potential of the OWL language. In fact, Rei rules knowledge is treated separately from OWL ontology knowledge due to its different syntactical form. OWL's inference is essentially considered as an oracle, i.e., the Rei policy engine treats inferences from OWL axioms as a virtual fact base. Hence, Rei rules can not be exploited in the reasoning process that infers new conclusions from the OWL's existing ontologies, which means that the Rei engine is able to reason about domain-specific knowledge, but not about policy specifications. As a main consequence of this limitation, Rei policy statements can not be classified by means of ontological reasoning. Therefore, in order to classify policies, the variables in the rules need to be instantiated, which reduces to a constraint satisfiability problem.

The reasoner for Rei 1.0, similar to the language, was entirely developed in Sicstus-Prolog considering its powerful reasoning capabilities [46]. However, in Rei 2.0 the whole design for the reasoner was shifted towards using XSB-Prolog [116], Flora [29, 118], and F-OWL [119]. XSB is a Logic Programming and Deductive Database system for Unix and Windows. It is being developed at the Computer Science Department of the Stony Brook University, in collaboration with Katholieke Universiteit Leuven, Universidade Nova de Lisboa, Uppsala Universitet and XSB, Inc. FLORA-2 is an advanced object-oriented knowledge base language and application development environment. The language of FLORA-2 is a dialect of F-logic with numerous extensions, including meta-programming in the style of HiLog and logical updates in the style of Transaction Logic. F-OWL is an ontology inference engine for the Web Ontology Language OWL. The ontology inference mechanism in F-OWL is
implemented using Flora-2 and XSB-Prolog.

As a result of using F-OWL, which is itself based on Flora-2 and XSB-Prolog, the underlying logic in the Rei framework can be considered a computational logic, having its roots in the deductive nature of XSB and F-Logic. Rei also has an integrated policy development environment to facilitate the definition of rules for Policy engineers which is called Ride [93].

2.4.4 KAoS (Knowledgeable Agent-Oriented System)

KAoS is a framework that provides policy and domain management services for agents and other distributed computing platforms [15, 45, 103, 102]. It has been deployed in a wide variety of multi-agent and distributed computing applications. KAoS policy services allow for the specification, management, conflict resolution and enforcement of policies within agent domains. KPAT, a powerful graphical user interface, allows non-specialists to specify and analyze complex policies without having to master the complexity of OWL. KAoS adopts an ontology-based approach for the purpose of semantic policy specification. In fact, policies are mainly represented in OWL as ontologies. The KAoS policy ontologies distinguish between authorizations and obligations. In KAoS, a policy constrains the actions that an agent is allowed or obliged to perform in a given context. In particular, each policy controls a well-defined action whose subject, target, and other context conditions are defined as property restrictions on the action type. Program 2.4 shows an example of KAoS authorization. The property performedBy is used to define the class to which the actor must belong for the policy to be satisfied.

In KAoS, context conditions that constrain a policy may be specified through the definition of appropriate classes defined via property restrictions. In particular, two main properties, i.e., the hasDataContext and the hasObjectContext properties, and their sub-properties are used to characterize the action context. Some sub-properties are defined in the KAoS ontology, like for instance the ones defining the actor (performedBy), the time and the target resource (accessedEntity) of an action, while others may be created within domain-specific ontologies. Program 2.5 shows the definition of a class, namely SkyTeamCustomer, which represents all the individuals that are flying with a company belonging to the Sky Team alliance. This class is defined as a subclass of the Customer
As these examples show, KAoS is based on an ontological approach to policy specification, which exploits OWL's description logic features to describe and specify policies and context conditions. In fact, contexts and related policies are expressed as ontologies. Therefore, KAoS is able to classify and reason about both domain and policy specification based on ontological subsumption, and to detect policy conflicts statically, i.e. at policy definition time.

However, a pure OWL approach encounters some difficulties with regard to the definition of some kinds of policies, specifically those requiring the use of variables. For instance, by relying purely on OWL, we could not define policies such as the FileSharing policy shown in Programs 2.1, 2.2, and 2.3; which defines constraints over property values that refer to statically unknown values, e.g., the policy owner location. Other examples include policies that contain parametric constraints, which are assigned a value only at the deployment or run time. For this reason, KAoS developers have introduced role-value maps as OWL extensions.
CHAPTER 2. POLICY MANAGEMENT AND WEB SERVICE TECHNOLOGIES

Program 2.5: A Class definition following KAoS conventions to be used by a KAoS policy and implemented them within the Java Theorem Prover (JTP), used by KAoS. The adoption of role value maps, description logic-based concept constructors that were originally introduced in the KL-ONE system, allows KAoS to specify constraints between property values expressed in OWL terms, and to define policy sets, i.e., groups of policies that share a common definition but can be singularly instantiated with different parameters. The proposed extensions effectively add sufficiently expressive flexibility to KAoS to represent the policies discussed in this paper. However, non-experienced users may have difficulties in writing and understanding these policies without the help of the KPAT graphical user interface. As Stanford’s JTP is a reasoner working over OWL (or DAML) knowledge bases, using this engine by KAoS makes it a framework based on the principals of description logic.

Table 2.1 gives a comparison of the three languages PeerTrust, Rei, and KAoS based on their reasoning engines, the knowledge representation methods, the application domain for these languages, etc.

2.4.5 eXtensible Access Control Markup Language (XACML)

eXtensible Access Control Markup Language (XACML) [73] is an XML specification for expressing policies for information access over the Internet and is being defined by the Organization for Advancement of Structured Information Standards (OASIS) technical committee. The language permits access control rules to be defined for securely browsing XML documents that can update individual document elements.
Similar to existing policy languages, XACML is used to specify subject-target-action-condition oriented policy in the context of a particular XML document. The notion of subject comprises identity, group, and role and the granularity of target objects is as fine as single elements within the document. The language supports roles, which are the same as groups, and are defined as collections of attributes relevant to a principal. XACML includes conditional authorization policies, as well as policies with external post-conditions to specify actions that must be executed prior to permitting an access.

As an example consider a policy that states "A person, identified by his or her patient number, may read any record for which he or she is the designated patient." This policy can be defined similar to Programs 2.6 and 2.7 in XACML.

Although XACML supports a fine granularity of access control specification, the policy is rather verbose and not really intended for human interpretation. XACML is intended to be used in conjunction with SAML (Security Assertion Markup Language [35]) assertions and messages, and can thus also be applied to certificate-based authorizations.

### 2.5 Web Rule Languages

Rules are among the most frequently used knowledge representation techniques. Figure 2.8 represents a hierarchy of rule types [10], from reaction rules (event-condition-action rules), via integrity rules (rules for consistency checking) and derivation rules (implicitly derived
<?xml version="1.0" encoding="UTF-8"?>
<Policy
xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xlns:md="http://www.med.example.com/schemas/record.xsd"
PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:1"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:denyoverrides">
  <PolicyDefaults>
    <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</XPathVersion>
  </PolicyDefaults>
  <Target/>
  <VariableDefinition VariableId="17590034">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <SubjectAttributeDesignator
          AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:patient-number"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Apply>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
      <AttributeSelector
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </Apply>
  </VariableDefinition>
  <Rule
    RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:1"
    Effect="Permit">
    <Description>
      A person may read any medical record in the
      http://www.med.example.com/schemas/record.xsd namespace
      for which he or she is the designated patient
    </Description>
  </Rule>
</Policy>

Program 2.6: A Sample Policy defined in XACML (Part I)
Program 2.7: A Sample Policy defined in XACML (Part2)
rules), to facts (derivation rules without premises).

Figure 2.8: The Hierarchy of rule types

A reaction rule typically includes an event which triggers the execution of the rule, conditions that are necessary to execute the action that the rule defines, the action itself, as well as pre- and post-conditions. An example of such a rule is the trigger rule in the SQL language (CREATE TRIGGER expression).

An integrity rule is generally a logical sentence. It specifies statements that must be true in all states, and defines the transitions of a discrete dynamic system for which the statements are defined. An example of such a rule is: “Confirmation of a booking for a car with a car rental agency must take into account possibility of existing booking/assignment for the car, the requested car type, and the requested date.” Well-known integrity rule languages are SQL and OCL.

A derivation rule consists of one or more conditions and a conclusion. Once the conditions are met, the rule is fired and the conclusion is executed. An example of such a rule (expressed in a computation-independent way) is “a car is available for rental if it is not assigned to any client and is not scheduled for service.” An example of a language for representing derivation rules is RuleML (cf. Section 2.5.1).

The main purpose of a rule markup language is to permit reuse, interchange, and publication of rules. Various rule markup languages are the vehicle for using rules on the Web and in other distributed systems [114]. They allow deploying, executing, publishing, and communicating rules in a network. In other words, rule markup languages allow us to specify business rules as modular, standalone, units in a declarative way, and to publish and interchange them between different systems and tools. They play an important role in facilitating business-to-customer (B2C) and business-to-business (B2B) interactions over the
Internet. Here we refer to two of the most known Web rule languages, namely RuleML and SWRL. REWERSE Rule Markup Language (R2ML) is another newly and rapidly emerging Web rule language that is the main technology we have used for defining our transformation framework. This Web rule language will be deeply discussed in Section 3.2.

### 2.5.1 RuleML

RuleML represents an initiative for creating a general rule markup language that will support different types of rules and different semantics [10]. It is conceptualized to capture the hierarchy of rule types shown in Figure 2.8. However, the current version of RuleML (February 2007) covers only some limited forms of rules. For example, it still does not have a general syntax for integrity and reaction rules [112].

RuleML is built on top of logic programming paradigm of first order logic (i.e. predicate logic). In the tradition of logic programming which is also followed by RuleML, research is focused on computable interpretations of predicate logic, by exploring a great number of semantic extensions and variations. OWL (as well as SWRL, cf. 2.5.2) stems from logic-based tradition of artificial intelligence where research is based on classical predicate logic (two-valued) as the one and the only logic.

An example of a RuleML rule that uses certain person’s attributes to define a rule of “hasMother” and “hasBrother” implies “hasUncle”, has been shown in Program 2.8.

Table 2.2 shows the basic tags in RuleML as well as their meanings.

### 2.5.2 Semantic Web Rule Language (SWRL)

The most recent initiative in the W3C consortium for rule languages is Semantic Web Rule Language (SWRL), which actually represents a combination of OWL and RuleML [41]. Like RDF and OWL, SWRL is based on classical first order logic. SWRL is restricted to OWL DL expressions, so it does not have support for high-level expressions that are allowed in RDF and OWL Full. This language is very similar to RuleML and its rules are of the form of an implication between an antecedent (body) and the consequent (head). The intended meaning can be read as “whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.” Both the antecedent (body) and the
Program 2.8: A sample rule in RuleML

Table 2.2: Basic Tags in RuleML

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Implies&gt;</td>
<td>An implication rule. It consists of a conclusion role (&lt;head&gt;) followed by a premise role (&lt;body&gt;), or equivalently (since roles constitute unordered elements), a premise role followed by a conclusion role</td>
</tr>
<tr>
<td>&lt;Atom&gt;</td>
<td>A logical atom, i.e. an expression formed from a predicate (or relation) applied to a collection of its (logical) arguments</td>
</tr>
<tr>
<td>&lt;Rel&gt;</td>
<td>A relation, i.e. a logical predicate, of an atom (&lt;Atom&gt;)</td>
</tr>
<tr>
<td>&lt;Var&gt;</td>
<td>A logical variable, as in logic programming</td>
</tr>
<tr>
<td>&lt;Ind&gt;</td>
<td>An individual constant, as in predicate logic, which can also be considered to be a fixed argument like RDF resources</td>
</tr>
<tr>
<td>&lt;op&gt;</td>
<td>An operator expression including either a relation (&lt;Rel&gt;) of an atom (&lt;Atom&gt;), a function name (&lt;Fun&gt;) of a (&lt;Expr&gt;), or a neutralized constant (&lt;Con&gt;) of a HiLog term (&lt;Hterm&gt;)</td>
</tr>
</tbody>
</table>
consequent (head) consist of zero or more atoms. Multiple atoms are connected with the conjunction operator.

Atoms in these rules can be of the form \( C(x) \), \( P(x, y) \), \( \text{sameAs}(x, y) \) or \( \text{differentFrom}(x, y) \), where \( C \) is an OWL description, \( P \) is an OWL property, and \( x, y \) are either variables, OWL individuals, or OWL data values. An example of such a rule for the implication \( \text{hasUncle} \) is shown in Program 2.9.

```xml
<ruleml:Implies>
  <ruleml:body>
    <swrlx:individualPropertyAtom swrlx:property="hasMother">
      <ruleml:Var>x1</ruleml:Var>
      <ruleml:Var>x2</ruleml:Var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="hasBrother">
      <ruleml:Var>x2</ruleml:Var>
      <ruleml:Var>x3</ruleml:Var>
    </swrlx:individualPropertyAtom>
  </ruleml:body>
  <ruleml:head>
    <swrlx:individualPropertyAtom swrlx:property="hasUncle">
      <ruleml:Var>x1</ruleml:Var>
      <ruleml:Var>x3</ruleml:Var>
    </swrlx:individualPropertyAtom>
  </ruleml:head>
</ruleml:Implies>
```

Program 2.9: Representation of Rule of Program 2.8 in SWRL

2.6 Summary

In this chapter we reviewed the concepts of Semantic Web, Semantic Web Services, Trust, and Policies. We showed in Figure 2.1 that trust is the topmost layer to be established in the Semantic layer cake (cf. Section 2.1). We also argued that employing the Semantic Web technology, and the Semantic Web languages, brings more flexibility and interoperability in defining and describing the context to which the policies should be applied. We showed how taking advantages in using ontologies for describing the to-be-protected context has been the primary approach in many policy languages (cf. Section 2.4).
We also reviewed the current issues with regards to establishing trust as well as the hard-policy and soft policy approaches devised to solved the issues. We also compared the differences between the soft- and hard-policy approaches (cf. Section 2.3).

In Section 2.2, we argued what Web services are and why it is useful to annotate the services with regards to some predefined ontologies. The benefits of this annotation were discussed as automatic discovery, orchestration, composition, and execution. Here we also add to the whole discussion about the Semantic Web, the Semantic Web services, and Trust that, for the services to be protected from misuse of malicious users, the web services should be well protected through using policies. This means, while we are defining the concepts of Semantic Web services, a room should be considered for policies, to be defined, and placed. This integration of policies and Web services, especially Semantic Web services, has been the point of attention in many research works [49, 74, 77, 104].

Interchanging policies among different parties is a part of the whole goal for enabling different parties with different Semantic Web services to communicate. Rules defined for specifying the behavior of Web services should be aligned with the policies defined to protect them, in order for the Web rule languages (cf. Section 2.5) to be able to provide consistent and accurate transformations between different services with different rule languages.
Chapter 3

Theoretical Foundations

In the previous chapter we discussed the need to interchange rules and policies between different parties, including service providers, requesting clients, and intelligent broker agents. This chapter is a dive into the technical and formal representation of our approach in using a rule interchange framework to define the transformations between the policy languages.

We start with introducing Rule Interchange Format (RIF) [31], a W3C initiative by the W3C RIF working group [106] for interchanging rules, in Section 3.1. In Section 3.2 we introduce the Web rule language used as the underlying technology to define the policy interchange between our chosen policy languages. As we mentioned in Section 2.5 this rule language, called the REWERSE Rule Markup Language (R2ML), is a newly emerging and rapidly evolving language to transform Web rules over the Internet. Section 3.2 will be a detailed description of the elements of the languages and the roles they can take as logical elements. Section 3.3 describes the theoretical foundation of our transformation framework as well as our chosen policy languages for the purpose of policy exchange through using R2ML.

We will later describe that the problem of transforming between different rule languages is not just a matter of providing one-to-one transformations between the elements of different policy languages. What make the transformations challenging are the differences in the underlying logic which is used to define the policies. Some of the policy languages - including Rei, PeerTrust, and PROTUNE - are following the principals of computational logic while some others such as KAoS follow the principals of description logic. This makes significant difference in defining the policies as the policy definition and evaluation procedure...
in description logic is looser than the way it is in computational logic (aka Logic programs). Section 3.3 describes the differences between different elements and constituents of the languages in each of the above logics and discusses how the mappings between these elements and accordingly their languages can happen.

3.1 Rule Interchange Format (RIF)

Rule-languages and rule-based systems have played seminal roles in the history of computer science and the evolution of information technology. From expert systems to deductive databases, the theory and practice of automating inference based on symbolic representations has had a rich history and continues to be a key technology driver. Due to the innovations made possible by the Internet, the World Wide Web, and, most recently, the Semantic Web, there is now even greater opportunity for growth in this sector. While some of these opportunities may require advances in research, others can be addressed by enabling existing rule-based technologies to interoperate according to standards-based methodologies and processes. The basic goal of the Rule Interchange Format (RIF) Working Group [106] is to devise such standards and make sure that they are not only useful in the current environment, but are easily extensible in order to deal with the evolution of rule technology and other enabling technologies.¹

3.1.1 The use cases for the RIF

The RIF working group has identified ten different use cases to justify the need for having a Rule Interchange Format. These use cases are intended to provide an ongoing reference point for the working group in its goal of providing a precise set of requirements for a RIF.

1. Negotiating eBusiness Contracts Across Rule Platforms


¹The content of this section, including the figures, has been taken from http://www.w3.org/TR/rif-ucr/. Copyright © 2006 W3C (MIT, ERCIM, Keio), All Rights Reserved. The document is available under the W3C Document License which grants permission to copy, and distribute the contents of this document in any medium for any purpose and without fee or royalty.
3. Collaborative Policy Development for Dynamic Spectrum Access

4. Access to Business Rules of Supply Chain Partners

5. Managing Inter-Organizational Business Policies and Practices

6. Ruleset Integration for Medical Decision Support

7. Interchanging Rule Extensions to OWL

8. Vocabulary Mapping for Data Integration

9. BPEL Orchestration of Rule-Based Web Services

10. Publishing Rules for Interlinked Metadata

Among the use cases above at least numbers 2, 3, and 5 explicitly determine a need to exchange policies between the interacting parties. Even number 4 can be considered as an interchange of policies between the supply chain partners, because Business Rules are mostly considered as a subset of policies [11].

The technical report published by the RIF working group [31] expresses its goals in the form of a critical factors analysis (CFA) diagram. This diagram identifies the Goals, Requirements, and Critical Success Factors and the relationships between these elements using the notions of Figure 3.1.

![Figure 3.1: The notions for the elements of a CFA diagram](image)

Using the CFA diagram, the goals of the RIF can be represented as in Figure 3.2

The primary goal of the RIF is to be an effective means of exchanging rules that has the potential to be widely adopted in industry and that is consistent with existing W3C
technologies and specifications. This can be further divided into the three subgoals of: 1) Exchangeability for rules, 2) Widescale adoption, and 3) W3C consistency, as shown in Figure 3.2.

The main goal of the RIF, among the three abovementioned subgoals, is to facilitate the exchange of rules. This mission is part of W3C’s larger goal of enabling the sharing of information in forms suited to machine processing:

- Rules themselves represent a valuable form of information for which there is not yet a standard interchange format, although significant progress has been made within the RuleML Initiative and elsewhere.

- Rules provide a powerful business logic representation, as business rules, in many modern information systems.

- Rules are often the technology of choice for creating maintainable adapters between information systems.

- As part of the Semantic Web architecture, rules can extend or complement the OWL Web Ontology Language to more thoroughly cover a broader set of applications, with knowledge being encoded in OWL or rules or both.

Looking at Figure 3.2, one can see six critical factors that have been identified by the RIF working group, among which coverage, extensibility, and predictability seem to be the most important ones. Also as it is shown in Figure 3.2, trying to provide better extensibility and coverage opposes low cost of implementation.
CHAPTER 3. THEORETICAL FOUNDATIONS

Figure 3.3: The CFA diagrams for the two critical factors of a) Coverage and b) Expandability

Coverage means providing a comprehensive framework to entail all the major classes of rule formalisms that are in widespread use, while Expandability means the possibility of the framework to evolve over time corresponding to the changes that happen to the currently existing policy languages. Figure 3.3a and 3.3b show the coverage and expandability CFA diagrams respectively, with the requirements for each critical factor represented in the diagrams. Finally, predictability entails the meaning that the RIF must be a sound basis for exchanging rules, i.e., it must be predictable what is exchanged when a ruleset is exchanged between partners and/or tools. Figure 3.4 shows the requirement for the predictability to be satisfied.

Figure 3.4: The RIF predictability requirement represented as a CFA diagram

Now that most of the requirements, critical factors, goals, and payoffs of a RIF language have been discussed and described, the aim will be designing a language that can admit to the requirements and the goals required for the RIF’s purpose. In the next section we introduce REWERSE Rule Markup Language (R2ML) as the language that we believe can
cover the requirements of the RIF and can address its goals.

3.2 REWERSE Rule Markup Language (R2ML)

It is expected that rule markup languages will be the primary driving force for the widespread use of rules both on the Web and in distributed systems. They allow for deploying, executing, publishing and communicating rules on the Web. They may also play the role of a lingua franca for exchanging rules between different systems and tools. They may be used, for example, to express derivation rules for enriching Web ontologies by adding definitions of derived concepts or for defining data access permissions; to describe and publish the reactive behavior of a system in the form of reaction rules; and to provide a complete XML-based specification of a software agent [113]. In a narrow sense, a rule markup language is concrete (XML-based) rule syntax for the Web. In a broader sense, it should have an abstract syntax as a common basis for defining various concrete languages serving different purposes. The main purpose of a rule markup language is to permit reuse, interchange and publication of rules.

REWVERSE Rule Markup Language (aka R2ML) project is part of the EU-funded REVERSE project, which follows the goal of providing a general rule markup language to make the deployment, execution, publishing, and communication of rules on the Web possible. The R2ML project was first initiated by Dr. Gerd Wagner ² and Dr. Adrian Giurca ³ from Institute of Informatics at Brandenburg University of Technology at Cottbus, Germany. Later Dr. Dragan Gâšević ⁴ also joined the project helping with evolving the R2ML project to the current version 0.5.

3.2.1 Designing The REWERSE Rule Markup Language

The approach chosen to develop the R2ML is based on the known Model Driven Architecture (MDA) [69] and Meta Object Facility (MOF) [78] defined by the Object Management Group (OMG) [70]. This means that the whole language definition of the R2ML language can be represented by using UML diagrams, as MOF uses UMLs graphical notation. The

²http://www.informatik.tu-cottbus.de/~gwagner
³http://www.informatik.tu-cottbus.de/~agiurca
⁴http://scis.athabascau.ca/scis/staff/index.jsp?ct=dragang&sn=faculty
To begin with, the rules have been considered in three different abstraction layers by the developers of R2ML (see Figure 3.5):

At the (computation-independent) business domain level (called CIM in OMG’s MDA), rules are statements that express (certain parts of) a business/domain policy (e.g., defining terms of the domain language or defining/constraining domain operations) in a declarative manner, typically using a natural language or a visual language. An example of such rules is: “The driver of a rental car must be at least 25 years old.”

At the platform-independent operational design level (called PIM in OMG’s MDA), rules are formal statements, expressed in some formalism or computational paradigm, which can be directly mapped to executable statements of a software system. Examples of rule languages at this level are SQL:1999 [96] and OCL 2.0 [115].

At the platform-specific implementation level (called PSM in OMG’s MDA), rules are statements in a language of a specific execution environment, such as Jess 3.4 [44] and XSB 2.6 Prolog [116].

R2ML tries to address all the requirements that have been recognized by RIF (see Section 3.1). As a Web rule language, R2ML considers four main types of rules, namely integrity
rules, derivation rules, reaction rules, and production rules. These types of rules will be reviewed in detail in Section 3.2.7. Below we start reviewing the elements and building blocks defined of R2ML mainly based on their UML metamodel.

3.2.2 R2ML's Vocabulary

R2ML provides a vocabulary that enables users to define their own domain in the form of objects and elements available in the domain of discourse. It helps the users to have a compliant representation of the knowledge about their respective domain in R2ML, which later can be exploited by the defined rules. The vocabulary consists of three sub-domains: Basic Content Vocabulary, Relational Content Vocabulary, and Functional Content Vocabulary.

R2ML’s Basic Content Vocabulary

The basic content vocabulary is mainly to define object names, object function names, data function names, noun concept names, and verb concept names. Among the abovementioned user-defined vocabularies, Object Names comprise Classes (or Object Types), which usually contain any object or object variable, and Data Types, which refer to some predefined data types. The concept of datatype in R2ML can be considered an extension of the datatype concept of RDF. Generally the datatypes and classes in R2ML are denoted using a URI reference.

Furthermore, user-defined verb concepts refer to properties, datatype predicates and associations. Properties are either attributes, if they are data-valued, or reference properties, if they are object-valued. A reference property corresponds to a non-literal-valued RDF property or to an OWL 'object property' and it is used in R2ML ReferencePropertyAtom. On the other hand, The R2ML datatype predicate accommodates the SWRL concept of built-in predicate. Both the reference property and datatype predicate are denoted by URIs in R2ML. R2ML also covers IndividualVariables and IndividualNames (aka constants).

In nutshell, the basic content vocabulary contains Vocabulary, VocabularyEntry, Predicate, Property, Type, DatatypePredicate, Attribute, Class, Datatype and ObjectName elements.
R2ML’s Relational Content Vocabulary

An association predicate or a generic predicate represents a relational content vocabulary in R2ML which is denoted by a URI reference. They allow to capture n-ary associations and association classes. Association predicates are used in association predicate atoms to connect an object to a class (or an object of that class).

R2ML’s Functional Content Vocabulary

An R2ML functor is a user-defined function that corresponds to the standard logic concept of a function. An R2ML data operation is a special type of a user-defined function that corresponds to a data-valued operation in a UML class model. An R2ML attribute is a special type of user-defined function that corresponds to a data-valued property in a UML class model. A user-defined object function is either a role function or an object operation. They are used in object terms. The elements of this part of R2ML’s vocabulary are EnumerationDatatype, GenericFunction, DatatypeFunction, Operation, DataOperation and ObjectOperation.

Figure 3.6 shows the UML representation of the whole vocabulary represented above.

3.2.3 R2ML’s Terms

The basic requirement in R2ML is to provide a means to define the objects, variables and basic concepts of a domain. These elements are defined in R2ML through using different terms. R2ML introduces three main types of terms: GenericTerm, ObjectTerm, and DataTerm. ObjectTerms are the superclass for defining variables, names, or operations which refer to the objects in a domain. DataTerms play the same role as ObjectTerms but they refer to data elements in a domain which are normally identified through using data literals that can be plain or typed. GenericTerm provides a higher level of abstraction in defining objects and variables. It captures the concept of predicate logic terms.

ReferencePropertyFunctionTerm is an object term that is used to model an association end. ObjectOperationTerm is an object term that is used to model an operation on contextual argument. Figure 3.7 shows the UML metamodel for an object term in R2ML.
CHAPTER 3. THEORETICAL FOUNDATIONS

Figure 3.6: The representation of R2ML's vocabulary in UML meta-model

Figure 3.7: The UML meta-model for R2ML's ObjectTerm
DataTerm is used to represent primitive data types and data values. As it can be seen in Figure 3.8, there are three types of data terms: DataVariable, DataLiteral, and DataFunctionTerm. DataVariable stands for a variable, DataLiteral represents a literal value, and finally there are three different types for DataFunctionTerm:

- **DatatypeFunctionTerm** represents arithmetic built-ins;
- **AttributeFunctionTerm** represents an attribute function (a function, which returns attribute value for an object);
- **DataOperationTerm** represents a user-defined function (method, for instance) and takes DataTerm or ObjectTerm as parameters.

Figure 3.8: The UML representation of R2ML's DataTerm

**GenericTerm** is used for modeling variables that may, or may not, have type (i.e. GenericVariable), constants (i.e. GenericEntityName), as well as generic functions (i.e. GenericFunctionTerm). Figure 3.9 shows the UML metamodel for the concept of GenericTerm in R2ML.

Looking back to Figures 3.7, 3.8, and 3.9, one can recognize three different types of variables which have been defined and used in R2ML, GenericVariables, ObjectVariables
and DataVariables. As a result, the UML representation for different types of variables can be shown similar to Figure 3.10.

Figure 3.10: The UML meta-model for R2ML variables

3.2.4 R2ML Atoms

Atoms are the building blocks of a rule. The Atoms of R2ML have been considered to be compatible with all the important concepts of OWL, SWRL, and RuleML. Figure 3.11 represents the UML metamodel for all of the defined atoms in R2ML.

An object description atom (Figure 3.12) refers to a class as a base type, and to zero or more classes as categories. It consists of a number of slots (attribute data slot and reference property object slot). An instance of such atom refers to one particular object.

An object classification atom refers to a class and consists of an object term (Figure 3.13).

An attribution atom consists of an object term as “subject”, and a data term as “value” (Figure 3.14).

A reference property atom (Figure 3.15) associates object terms as “subjects” with
Figure 3.11: The UML representation for R2ML Atoms

Figure 3.12: The UML representation of R2ML's ObjectDescriptionAtom

Figure 3.13: The UML meta-model for ObjectClassificationAtom
In order to directly support common fact types of natural language, it is important to have n-ary predicates (for n > 2).

An association atom (Figure 3.16) is constructed using an n-ary predicate as association predicate, a collection of data terms as "data arguments", and a collection of object terms as "object arguments".

Both equality atom and inequality atom (Figure 3.17) are composed of two or more object terms.

A data classification atom (Figure 3.18) consists of a data term and refers to a data type.
CHAPTER 3. THEORETICAL FOUNDATIONS

3.2.5 R2ML Formulas

R2ML provides two abstract concepts for formulas: the concept of AndOrNafNegFormula (Figure 3.20), which represents the most general quantifier-free logical formula with weak and strong negations, and the concept of LogicalFormula (Figure 3.21), which corresponds to a general first order formula.
R2ML supports two kinds of negation (as shown in Figure 3.30). The distinction between weak and strong negation is used in several computational languages (like SQL [96]), and it is presented in [110]. Weak negation captures the absence of positive information, while strong negation captures the presence of explicit negative information. Weak negation captures the computational concept of negation-as-failure (or closed-world negation).

3.2.6 R2ML Actions

R2ML supports five types of actions: *InvokeAction*, *AssignAction*, *CreateAction*, *DeleteAction* and *SOAPAction* (Figure 3.22).
InvokeAction refers to an UML Operation and contains a list of arguments. This action invokes an operation with a list of arguments. AssignAction refers to an UML Property and contains a DataTerm. This action assigns a value to a property. CreateAction refers to an UML Class and to an UML InputPin as a parameter. DeleteAction refers to an UML Class and contains an Object Variable. This action removes an instance of the Class. SOAPAction refers to the way that certain Web service is called with SOAP message.

3.2.7 R2ML Rules

Having defined all of the known and required concepts of R2ML (cf. Sections 3.2.1 to 3.2.6) to cover the available concepts on the Web, the developers of R2ML have introduced four main types of rules, namely integrity, derivation, production, and reaction rules, as the main classes to be considered for Web rules. This section deeply describes these types of rules.

Integrity Rules

Integrity rules in R2ML, also known as (integrity) constraints, consist of a constraint assertion, which is a sentence in a logical language such as first-order predicate logic or OCL (Figure 3.23). R2ML framework supports two kinds of integrity rules: alethic and deontic integrity rules. An alethic integrity rule can be expressed with a phrase, such as “it is necessarily the case that”, whereas a deontic one can be expressed with phrases, such as “it
is obligatory that” or “it should be the case that”.

Constraint assertion is a logical sentence that must necessarily (i.e. alethic), or that should (i.e. deontic), hold in all evolving states and state transition histories of the discrete dynamic system to which it applies. An integrity rule cannot have free variables, i.e. all variables in the formula are quantified.

An example of integrity rule on CIM level is: “if rental is not a one way rental then return branch of rental must be the same as pick-up branch of rental”. This rule in R2ML XML concrete syntax is shown in Program 3.1

**Derivation Rules**

Derivation Rules in R2ML, have “conditions” and “conclusions” (Figure 3.24a). In R2ML language the conditions of a derivation rule are \textit{AndOrNafNegFormula}. Conclusions are restricted to a literal conjunction of atoms as shown in Figure 3.24b.

An example of derivation rule on CIM level is: “the discount for a customer buying a product is 7.5 percent if the customer is premium customer and the product is luxury”. This rule in R2ML XML concrete syntax is shown in Program 3.2.

**Production Rules**

Production rules in R2ML consist of “conditions” and “post-conditions” (Figure 3.25). The conditions and post-conditions of a production rule are \textit{AndOrNafNegFormulas}. A production rule may execute an R2ML Action (see Section 3.2.6). An example of production rule on CIM level is: “if the order value is greater than 1000 and the customer type is not gold then give a 10% discount”. This rule in R2ML XML concrete syntax is shown in Program 3.3.
Program 3.1: A sample integrity rule in R2ML XML

Figure 3.24: The UML meta-model for R2ML's derivation rule
Figure 3.25: The UML meta-model for R2ML’s production rule

```
<r2ml:DerivationRule r2ml:id="DR004">
  <r2ml:conditions>
    <r2ml:ObjectClassificationAtom r2ml:classificationPredicateID="buy">
      <r2ml:objectArguments>
        <r2ml:objectVariable r2ml:name="customer"/>
        <r2ml:objectVariable r2ml:name="product"/>
      </r2ml:objectArguments>
    </r2ml:ObjectClassificationAtom>
  </r2ml:conditions>
  <r2ml:conclusion>
    <r2ml:AttributionAtom r2ml:attributeID="discount">
      <r2ml:subject>
        <r2ml:objectVariable r2ml:name="customer"/>
      </r2ml:subject>
      <r2ml:value>
        <r2ml:TypedLiteral r2ml:datatype="xs:decimal" r2ml:lexicalValue="7.5"/>
      </r2ml:value>
    </r2ml:AttributionAtom>
  </r2ml:conclusion>
</r2ml:DerivationRule>
```

Program 3.2: A sample derivation rule in R2ML XML
CHAPTER 3. THEORETICAL FOUNDATIONS

Program 3.3: A sample production rule in R2ML XML

```xml
<r2ml:ProductionRule r2ml:id="PR001" >
  <r2ml:conditions>
    <r2ml:qf.Conjunction>
      <r2ml:DataPredicateAtom
        r2ml:dataPredicateID="swrlb:greaterThan">
        <r2ml:dataArguments>
          <r2ml:AttributeFunctionTerm
            r2ml:attributeID="orderValue">
            <r2ml:contextArgument>
              <r2ml:ObjectVariable r2ml:name="order"
                r2ml:classID="srv:Order"/>
            </r2ml:contextArgument>
          </r2ml:AttributeFunctionTerm>
          <r2ml:TypedLiteral r2ml:lexicalValue="1000"
            r2ml:type="xs:positiveInteger"/>
        </r2ml:dataArguments>
      </r2ml:DataPredicateAtom>
      <r2ml:DataPredicateAtom
        r2ml:dataPredicateID="swrlb:equal" r2ml:isNegated="true">
        <r2ml:dataArguments>
          <r2ml:AttributeFunctionTerm
            r2ml:attributeID="customerRating">
            <r2ml:contextArgument>
              <r2ml:ObjectVariable r2ml:name="order"
                r2ml:classID="srv:Order"/>
            </r2ml:contextArgument>
          </r2ml:AttributeFunctionTerm>
          <r2ml:TypedLiteral r2ml:lexicalValue="gold"
            r2ml:type="xs:string"/>
        </r2ml:dataArguments>
      </r2ml:DataPredicateAtom>
    </r2ml:qf.Conjunction>
  </r2ml:conditions>
  <r2ml:producedAction>
    <r2ml:AssignActionExpression r2ml:propertyID="srv:discount">
      <r2ml:contextArgument>
        <r2ml:ObjectVariable r2ml:name="order"
          r2ml:classID="srv:Order"/>
      </r2ml:contextArgument>
      <r2ml:TypedLiteral r2ml:lexicalValue="10"
        r2ml:type="xs:positiveInteger"/>
    </r2ml:AssignActionExpression>
  </r2ml:producedAction>
</r2ml:ProductionRule>
```
Reaction Rules

Reaction rules consist of a mandatory triggering event expression, an optional condition, and a produced action or a post-condition (or both), which are roles of type \textit{EventExpression}, \textit{AndOrNafNegFormula}, \textit{ActionExpression}, and \textit{AndOrNafNegFormula}, respectively, as shown in Figure 3.26. While the condition of a reaction rule is exactly like the condition of a derivation rule, i.e. a quantifier-free formula, the post-condition is restricted to a conjunction of possibly negated atoms. There are two types of reaction rules: those that do not have a post-condition, which are the well-known \textit{Event-Condition-Action (ECA)} rules, and those that do have a postcondition.

![Figure 3.26: The UML meta-model for R2ML's reaction rules](image)

A reaction rule consists of a \textit{triggeringEvent} which is an R2ML \textit{EventExpression}, that is either atomic or composite. A reaction rule also has \textit{conditions} which are represented as a collection of \textit{AndOrNafNegFormula} and \textit{postconditions}. Finally, a reaction rule has an element called \textit{producedAction} which is an R2ML action, that represents a condition change in a system. R2ML v0.5 is also capable of defining composite actions such as sequential or parallel actions.

An example of reaction rule on CIM level is: \textit{if customer returns a car and the car has more than 5000km from the last service then send the car to the service}. This rule in R2ML XML concrete syntax is shown in Program 3.4.

3.3 Theoretical Foundations of Transforming Policies

In Sections 2.4.3 and 2.4.4, we studied the structure and the properties of the two policy languages, Rei and KAoS, besides the other policy languages that we mentioned in Section 2.4. To provide a policy interchange format, we had to start with at least two policy languages.
Program 3.4: A sample reaction rule in R2ML XML
CHAPTER 3. THEORETICAL FOUNDATIONS

We considered KAoS and Rei as the two most suitable policy languages to begin with. One reason to choose KAoS and Rei was that they both use XML syntax in representing their rules. This would ease the process of developing the transformations for these two policy languages. Furthermore, as most of the current policy languages including KAoS and Rei have got the initial ideas from Ponder\(^5\) [23], there are some similarities in the way these two policy languages refer to different policies and concepts in a domain of discourse.

However, besides all the similarities between Rei and KAoS, there is one major difference between these two policy languages. Rei, as mentioned in Section 2.4.3, has been developed based on XSB-Prolog [116], Flora [29, 118], and F-OWL [119], while KAoS has been developed based on Stanford's Java Theorem Prover (JTP) [83]. The main difference between these technologies is that XSB-Prolog and Flora follow the concepts of computational logic (aka Logic Programs) while Stanford's JTP is based on description logic [63]. Establishing a framework that can cover both logics, guarantees the coverage of a wide range of existing policy languages and makes our policy language functional over various domains. This is the main argument over choosing Rei and KAoS to build the policy interchange framework.

Next, we provide a comparison of these two languages as well as describing Description Logic versus Computational Logic, in order to be able to provide accurate mappings that not only can cover the syntactical similarities, but also can provide clear transformations in the level of logics as well. Interchanging the policies is a critical task as the interchanged policies should have the same effects as the original ones. Thus, there is a need to provide clear logical models for the policy domain in order to provide accurate semantic mappings between the similar concepts of the policy languages.

3.3.1 Comparing KAoS and Rei

Although the traditional policy languages were mainly concerned about the users of a domain, the recently developed Web policy languages pay special attention to the context of communication. It is obvious that in pervasive environments, like the Web, the context is more stable than the users and thus putting constraints over the context results in a better

\(^5\)Ponder is a java-based Object Oriented policy language to specify management and security policies for distributed systems. More about Ponder can be found at http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml
As we discussed in Section 2.4, Rei and KAoS are two semantically enriched Web policy languages that give special care to the context of interactions. Using Semantic Web ontologies to define the resources, the behavior, and the users of a domain enables these two languages to easily adjust themselves to the target system regardless of the number of resources and users in act. KAoS describes the entities and concepts of its world using OWL building blocks. Similarly Rei can understand and reason over a domain of concepts defined in either RDF or OWL.

In terms of available policy rules both KAoS and Rei have four main types. Permission, Prohibition, Obligation, and Dispensation in Rei are respectively equivalent to PosAuthorizationPolicy, NegAuthorizationPolicy, PosObligationPolicy, and NegObligationPolicy in KAoS. In both of the languages these elements are instantiated from classes that correspond to the concepts for authorization and obligation with appropriate values allocated to their properties for the users and actions to be controlled. The defined policy rules in each of the languages are then sent to a reasoner that performs the process of conflict resolution and decision making for the rules that match the current state of the world. We described in Sections 2.4.3 and 2.4.4 that this task is done by using Stanford’s Java Theorem Prover (JTP) in KAoS and a Prolog engine in Rei.

The main difference between KAoS and Rei is the logic of expressing the rules. KAoS follows description logic to define policies and all the policy rules for KAoS are defined in the form of OWL expressions. On the other hand, Rei uses its own language that defines rules in terms of Prolog predicates expressed as RDF triples. The Rei language syntactically follows the conventions of RDF but it semantically resembles Prolog predicates. Admitting to the semantics of Prolog as a logic programming language has made Rei a language based on declarative logic. Rei refers to the policy rules as deontic elements, stemming from deontic logic [66, 79], which have properties to define the actor, the action, and the constraints of a rule. Rei also allows multi-level definition of constraints and actors (e.g. actors for a policy set and actors for a policy rule).
The process of rule enforcement in KAoS is done by extending its enforcement engine depending on the domain it is going to be used in. In Rei, the main deficiency is non-existence of any rule enforcement engine to assure the execution of the policies. Yet, due to the more deterministic properties of declarative logic, the process of reasoning over the dynamically determined values in Rei policies is more accurate than KAoS in which chances of dealing with unknown situations are likely to happen.

In order for processes and services to communicate remotely, Rei relies on a rich set of Speech Acts. Speech Acts are used to express six main concepts: Delegation, Revocation, Request, Cancel, Command, and Promise. Having the values for sender and receiver defined in each of these speech acts, it demonstrates a request from a sender to a receiver to perform one of the above actions. Conversely, in KAoS the remote communication procedure is done through the message passing of the underlying platform.

Different quantifying expressions, inheritance relationships, cardinality restrictions, and properties of such can be explicitly expressed in KAoS, thanks to the constructs of OWL. This also enables KAoS to perform static conflict resolution and also policy disclosure. KAoS has its classes and properties already defined in OWL ontologies, referred to as KAoS Policy Ontologies (KPO) [99], which are accessible in [60]. Analogous to KAoS, a policy in Rei is instantiated from the policy class defined in the set of Rei ontologies. The policy instance in Rei guides the behavior of entities in the policy domain [88].

3.3.2 Description Logic vs. Computational Logic

Referring to Section 3.3.1, we argued that the problem of providing transformations between KAoS and Rei expands to the problem of bridging between the descriptive logic world and its computational logic counterpart, due to following descriptive and computational logic principals by these two policy languages respectively. It is important for R2ML because we need R2ML to serve as a conductor between the two worlds. R2ML has been designed having the properties of both open world (i.e. descriptive logic) and close world (i.e. computational logic) in mind. Knowing the formal method of providing transformations from computational logic to descriptive logic and back would help in providing more meaningful transformations with less information loss.
[36] introduces an elaborate approach on mapping the basic elements of description logic to computational logic (referred to as Logic Programs (LP) in [36]). The work of [36] considers Semantic Web Services (see Section 2.2) as a task-oriented motivation for combining rules with ontologies, which is exactly aligned with our purpose of providing transformations between rules and policies; i.e., to better enable discovery, composition, orchestration, and execution of the Web services with various types of policies and policy languages.

Figure 3.27: Expressive overlap of Description Logic and Logic Programs and where Rei and KAoS sit in this classification

Description Logic is a subset of the well-known First Order Logic (FOL) without function symbols [36], which is similar to a fragment of Horn FOL, def-Horn, that also contains no function symbols, both have been shown in Figure 3.27. LP, on the other hand, intersects with FOL but neither includes nor is fully included by FOL, e.g., FOL can express positive disjunctions, which are inexpressible in LP. Furthermore, several expressive features of LP, which are frequently used in practical rule-based applications, are inexpressible in FOL (e.g., Negation-as-Failure). Consequently, while providing mappings from a DL-based language to a LP-based one, there is no possibility to cover all the concepts. Yet, most of the concepts of these languages are transformable from one to another due to the similarities between DL and def-Horn shown as the intersection of these two sets in Figure 3.27. Table 3.1 shows some of the concepts in DL and their equivalences in FOL (see [36] for a complete list of the mappings).
Table 3.1: DL syntax, FOL representation, and the corresponding DAML+OIL constructors

<table>
<thead>
<tr>
<th>DL</th>
<th>FOL</th>
<th>Example</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a : C$</td>
<td>$C(a)$</td>
<td>jim : Human</td>
<td>element0f</td>
</tr>
<tr>
<td>$(a,b) : P$</td>
<td>$P(a,b)$</td>
<td>(jim,lifestyle) : Royal</td>
<td>hasProperty</td>
</tr>
<tr>
<td>$C \subseteq D$</td>
<td>$\forall x . C(x) \rightarrow D(x)$</td>
<td>Human $\subseteq$ Animal $\cap$ Biped</td>
<td>subClass0f</td>
</tr>
<tr>
<td>$Q \subseteq P$</td>
<td>$\forall x,y . Q(x,y) \rightarrow P(x,y)$</td>
<td>Royal $\subseteq$ Wealthy</td>
<td>subProperty0f</td>
</tr>
<tr>
<td>$C_1 \cap \ldots \cap C_n$</td>
<td>$C_1(x) \land \ldots \land C_n(x)$</td>
<td>Animal $\cap$ Biped</td>
<td>intersection0f</td>
</tr>
<tr>
<td>$C_1 \cup \ldots \cup C_n$</td>
<td>$C_1(x) \lor \ldots \lor C_n(x)$</td>
<td>Doctor $\lor$ Lawyer</td>
<td>union0f</td>
</tr>
<tr>
<td>$\neg C$</td>
<td>$\neg C(x)$</td>
<td>$\neg$Male</td>
<td>complement0f</td>
</tr>
<tr>
<td>$\exists P . C$</td>
<td>$\exists y . (P(x,y) \land C(y))$</td>
<td>$\exists$hasChild.Lawyer</td>
<td>hasClass</td>
</tr>
<tr>
<td>$\forall P . C$</td>
<td>$\forall y . (P(x,y) \rightarrow C(y))$</td>
<td>$\forall$hasChild.Doctor</td>
<td>toClass</td>
</tr>
</tbody>
</table>

OWL as a subset of RDF(S) corresponds to a fragment of classical FOL. It has been argued in [36] that OWL elements are convertible to definite Horn FOL elements which in turn are convertible to definite Datalog Logic Programs as a restricted model of Logic Programs (LPs). Establishing precise mappings between the concepts of Logic Programs and Description Logic, according to the shared concepts of these two logics (referred to as Description Logic Programs in [36] - see Figure 3.27), makes transformation of the concepts between these two worlds possible and enables sharing of the information between their corresponding systems more conveniently. Here we review a number of mappings and leave the rest to the reader by referring him/her to [36].

- A DL inclusion axiom corresponds to a FOL implication. This leads to a straightforward mapping from class and property inclusion axioms to def-Horn rules as follows:

$$C \subseteq D,$$ i.e. class $C$ is a subclass of class $D$, maps to:

$$D(x) \leftarrow C(x)$$

Referring to Table 3.1 and the provided example in the third and the fifth rows of this table, the above mapping can be stated in the form of Propositional Logic as: "being Human implies being Animal and Biped."

$$Q \subseteq P,$$ i.e. $Q$ is a subproperty of $P$, maps to:

$$P(x,y) \leftarrow Q(x,y)$$
Again, referring to the fourth row of Table 3.1, the above mapping can be stated in the form of Propositional Logic as: "having a **Royal lifestyle** implies having a **Wealthy lifestyle**."

- Asserted class-instance (type) and instance-property-instance relationships, which correspond to DL axioms of the form $a : C$ and $(a, b) : P$ respectively, are equivalent to FOL sentences of the form $C(a)$ and $P(a, b)$, where $a$ and $b$ are constants; so we have:

  $$a : C, \text{ i.e. the individual } a \text{ is an instance of the class } C \text{ maps to: } C(a)$$

  $$\langle a, b \rangle : P, \text{ i.e. the individual } a \text{ is related to individual } b \text{ via the property } P \text{ maps to: } P(a, b)$$

- Conjunction can be directly expressed in the body of def-Horn rule. E.g. when a conjunction occurs on the left hand side of a subclass axiom, it simply becomes conjunction in the body of the corresponding rule:

  $$C_1 \cap C_2 \subseteq D \quad \equiv \quad D(x) \leftarrow C_1(x) \land C_2(x)$$

  Similarly, when a conjunction occurs on the right hand side of a subclass axiom, it becomes conjunction in the head of the corresponding rule:

  $$C \subseteq D_1 \cap D_2 \quad \equiv \quad D_1(x) \land D_2(x) \leftarrow C(x)$$

  which is then easily transformed into a pair of def-Horn rules as below:

  $$D_1(x) \leftarrow C(x)$$

  $$D_2(x) \leftarrow C(x)$$

- A DL class can be formed from a disjunction of existing classes. When a disjunction occurs on the left hand side of a unary subclass axiom, it simply becomes disjunction in the body of the corresponding rule:

  $$C_1 \cup C_2 \subseteq D \quad \equiv \quad D(x) \leftarrow C_1(x) \lor C_2(x)$$

  This again can be easily transformed into a pair of def-Horn rules:
CHAPTER 3. THEORETICAL FOUNDATIONS

- When a disjunction occurs on the right hand side of a subclass axiom, it becomes a disjunction in the head of the corresponding rule, and this cannot be handled within the def-Horn framework.

- In a DL, the universal quantifier can only be used in restrictions- expressions of the form $\forall P.C$. This is equivalent to a FOL clause of the form $\forall y. P(x, y) \rightarrow C(y)$. $P$ must be a single primitive property, but $C$ may be a compound expression. Therefore, when a universal restriction occurs on the right hand side of a subclass axiom, it becomes an implication in the head of the corresponding rule:

\[
C \subseteq \forall P.D \equiv (D(y) \leftarrow P(x, y)) \leftarrow C(x)
\]

which is easily transformed into the standard def-Horn rule:

\[
D(y) \leftarrow P(x, y) \land C(x)
\]

- When a universal restriction occurs on the left hand side of a subclass axiom, it becomes an implication in the body of the corresponding rule. This cannot, in general, be mapped into def-Horn as it would require negation in a rule body.

- In DL, the existential quantifier (like the universal quantifier) can only be used in restrictions of the form $\exists P.C$. This is equivalent to a FOL clause of the form $\exists y. P(x, y) \land C(y)$. $P$ must be a single primitive property, but $C$ may be a compound expression. When an existential restriction occurs on the left hand side of a subclass axiom, it becomes a conjunction in the body of a standard def-Horn rule:

\[
\exists P.C \subseteq D \equiv D(x) \leftarrow P(x, y) \land C(y)
\]

- When an existential restriction occurs on the right hand side of a subclass axiom, it becomes a conjunction in the head of the corresponding rule, with a variable that is existentially quantified. This cannot be handled within the def-Horn framework.

In the next chapter we show how the defined mapping rules of Table 3.1 will be used to provide transformations between KAoS and Rei.
3.4 Summary

In this chapter, we discussed the theoretical foundation of our framework according to the available technologies and languages for exchanging the rules over the Internet. In Section 3.1, we showed the main purpose of defining a Rule Interchange Format (RIF) to cover the concepts of all types of rules available on the Web. We argued how among all the use cases, defined for RIF, at least three of them pay special attention to the concepts of policies. We discussed how sharing the policies among the systems, with similar interests on one domain, can enable them to communicate and exchange the information.

In Section 3.2, we introduced REWERSE Rule Markup Language (R2ML), as one of the two major submissions for RIF initiative (beside RuleML). We elaborated on how R2ML deals with rules, what types of rules are defined in R2ML, how these rule types are expressive enough to cover the concepts of all types of rules on the Web, etc. We also introduced the building blocks of the R2ML language, namely Atoms, Formulas, Terms, and Actions, along with their use cases and meanings. Considering all the characteristics of R2ML, we came up with the decision that using R2ML for the purpose of information exchange can be the appropriate way of taking the early steps in addressing the use cases and the requirements of the RIF and moving towards its ultimate goal of information exchange. It becomes more important once we notice that, so far, there has been no practical effort in transforming between policy languages and our approach can be considered the first attempt in this area.

In Section 3.3, we compared KAoS and Rei and described the underlying logic of these two languages, i.e. computational logic for Rei and description logic for KAoS. This mainly helps with better being able to provide mappings between the concepts of these two languages as we will later refer to it in Section 4.3.

Having all the above information about Rei, KAoS, R2ML, and the underlying logic of them, we are now ready to provide transformations between these languages which will be the main focus in the next chapters of this dissertation.
Chapter 4

The Policy Interchange Framework

This chapter has been fully devoted to describing the semantic models of Rei and KAoS that leads to Policy Interchange Framework. In this chapter, we also define the abstract models for the transformations from KAoS and Rei to R2ML, and then back from R2ML to either KAoS or Rei. We will show how the logical differences between KAoS and Rei will be covered by using the mapping rules of Section 3.3.2 about description and computational logics.

The first section in this chapter, Section 4.1 explains the layered architecture for our Policy Interchange Framework. In this section we discuss the issues that should be considered towards developing a general policy architecture in order to make it flexible enough to cope with the concepts from different policy languages. We will also discuss the reasons and advantages behind adopting a layered architecture for our policy framework.

Section 4.2 provides metamodels for the policy languages we have used in our framework, i.e. KAoS and Rei, in terms of UML diagrams. It enables us to have better understanding of the required elements in the core of the policy framework. This also enables the users who are not familiar with the XML syntax of these languages to better understand the concepts and elements of each language.

Section 4.3.1 describes our mapping rules from KAoS policy language to R2ML. Referring to our metamodel definition of Rei and KAoS (Section 4.2), we show how the identified relations between different concepts of these policy languages will help with providing transformations between these two languages. The mappings of the concepts of either of
the policy languages will be first formally shown and matched to the concepts of our rule language, namely R2ML. Then the transformations will be described in detail.

Section 4.3.2 will follow a similar structure to Section 4.3.1 and again the concepts between the Rei policy language and the rule language will be matched, compared, and the raised problems will be resolved. The most important matter in our effort in providing mappings from Rei or KAoS and back, through using R2ML, was to make sure that the obtained R2ML rule set from each policy language is as close as possible to the R2ML rule set obtained from the other policy language. Since Rei and KAoS refer to different authentication and authorization concepts using different elements of their language, which can be possibly based on different logic and ideology, chances are that a direct mapping from one policy language to R2ML would be inconsistent to the results of transforming the other policy language to R2ML. Providing compliant pieces of R2ML from either Rei or KAoS can be considered as a significant improvement in our method, as compared to the other transformations from other rule languages to R2ML [67].

Finally, Section 4.3.3 mainly discusses our chosen method of transforming the obtained R2ML file to either of these two languages. Again, providing one single conceptual definition for our derived R2ML rule sets exempts us of the need to provide mapping between different R2ML excerpts which is a cumbersome and auxiliary task.

4.1 Layered Policy Interchange Framework

Interchanging the rules in general, and the policies in particular, between different business enterprises is a goal followed by RIF working group [31]. We already discussed, in Section 3.1.1, why interchanging the policies between different business partners is important to achieve. Nonetheless, the variety of policy languages that have been developed so far, the lack of a standard for defining policies, and the limited number of experts in each of the existing policy languages, made us think about designing a policy interchange framework that can easily expand to cope with different policy languages, especially the ones that have not been developed yet. To do so, we tried to think about what is needed by a policy framework to easily support expansion and integration of different policy languages.
It is worth noting that while interchanging the policies, we deal with their logic, their abstract syntax, their concrete syntax, and also the semantics that each term in each language carries. Our studies led us to a point where we realized that, in order to have such a framework, we need to start from the logic that the language follows, to the abstract syntax that it represents, having the semantics of its concepts in mind. The concrete syntax of the languages is probably the least important issue to deal with, as there are powerful transformation tools and languages, such as QVT/ATL [85] and XSLT [107], that can mine through the concrete definition of the languages and extract the appropriate concepts.

For the abstract syntax of the language, we normally encounter the general concepts that are shared between the languages with similar purposes, and the language-specific concepts that are not shared between all the languages of one single category. As an example let us compare Java and C++. For Java, as a third generation general purpose language, we have object oriented concepts for concrete and abstract class definitions, inheritence, attribute visibility, and etc. Similarly C++ neatly covers all these concepts which are common between all languages that support Object Oriented Programming. Beside all the similarities that these two languages share, there are concepts that are not covered by both languages. The simplest one is package. In Java, all the classes are defined relative to a package, even if it is not explicitly indicated. However C++ does not support packaging. The concepts such as inheritence, class definition, etc., are general concepts shared between Java and C++, while supporting or not supporting concepts such as package makes them language specific.

The language-specific concepts take more important roles in languages that are in a lower level of abstraction. Policy languages, due to their characteristics in addressing low level domain specific concepts, and the differences in the domains that they need to be deployed in, may have various dissimilar concepts that are specific for each language. A powerful policy interchange framework should be able to clearly distinguish between the logic, the language-independent, and the language-specific concepts and try to have general definitions for as many of these concepts as possible.

---

1It should be noted that packaging is a known object oriented approach to provide modularized programming. In this respect, not supporting the concept package by C++ makes this language specific as compared to its other counterparts such as Java.
Figure 4.1: The Layered Policy Interchange Framework Architecture

Figure 4.1 shows our proposed architecture for a policy interchange framework. As the figure shows, the policy framework, first starts with identifying the similarities and dissimilarities between the underlying logics of the policy languages. The two main logics that are widely used in defining rules and representing the knowledge of a domain are computational logic and descriptive logic (cf. Section 3.3.2) and there has been a lot of research on how to map the concepts of these two logics [36, 59]. Our policy framework will exploit the work that has been already done in this area by different researchers and logicians all around the world.

The next level in the architecture is to identify the general concepts that are common between the policy languages. Once these concepts are recognized, the rules from one source language to one target language can be mapped to some degree, having the vocabulary of the two languages identified and mapped. For the language-specific concepts of a source policy language, either it is possible to define the concepts through using a series of concepts from the target language, or simply the concepts are not convertible. Depending on the importance of the meanings that the un-mappable concepts carry, the transformation to the target policy language would be successful or unsuccessful.

As Figure 4.1 shows, by moving to the upper levels of the architecture, the generality of the mappings is significantly decreased, such that reusing it for other policy languages becomes impossible in the top most level. Nonetheless, it is possible for the concepts in the lower levels to be shared between different policy languages.

To plug a new policy language to the framework, first and foremost, the underlying logic
of the policy language should be identified and mapping rules to cover the corresponding concepts should be developed. Moving to the upper level in the architecture, the common policy concepts are identified and mapped to the concepts of the general policy model used. For the topmost level, a detailed review of the source policy language by its experts is required to work around the concepts that are not mappable.

In the following sections of this chapter, we try to extract the abstract syntax of KAoS and Rei by identifying their metamodels, extract the language-independent and languagespecific concepts that these two policy languages cover, and argue how the mapping between these policy languages can happen without facing serious information loss.

4.2 MetaModels for Rei and KAoS

Concept modeling is considered a critical step in better understanding and comprehending the constituents of a system in general, and a language in particular. In this section we present the UML metamodels for KAoS and Rei to show the concepts and commonalities of the languages on one hand, and to be able to find the dissimilarities between them on the other hand.

As we mentioned earlier, both KAoS and Rei exploit the use of Web Ontology Language (OWL) to define their concepts. Thus providing a meta-model for the languages can be done through transforming the OWL definition of the languages to a UML metamodel. There have been several efforts in defining standard mappings from ontology languages to UML. In particular, the Ontology Definition Metamodel (ODM) [76] initiative follows the goal of using OMG’s Meta Object Facility (MOF) as a metamodeling language to develop a linguistic metamodel to represent the ontology languages. Although ODM, itself, is still undergoing modifications, we chose to use the current state of ODM to represent Rei and KAoS. However, at some points, based on the needs and the real meaning of the concepts in each language, we had to slightly modify the ODM definition for getting a better reflection of the concepts through using UML constructs.
4.2.1 Modular Policy Languages

Rei and KAoS, both, are highly modular policy languages consisting of several modules, referred to as *ontologies* in OWL and *packages* in UML. Figure 4.2 shows the defined modules for KAoS and Figure 4.3 shows the defined modules for Rei, each represented as UML packages standing for their corresponding ontologies.

As one can see, there are a lot of packages and relations between the packages for KAoS policy language, but there are fewer packages with no relations between them in Rei. That is to say, none of the packages in Rei includes the others, which seems somehow awkward in a multi-modular approach. Instead, Rei redefines the classes and entities which have been defined in the other packages, using DTD’s entity definition and OWL’s rdf:about, as a method to provide connections between different modules (see Program 4.1). Furthermore, looking at the top-most layer of language definitions for both KAoS and Rei, it is easily
perceivable that KAoS is a much more extensive language as compared to Rei.

4.2.2 Policies for KAoS and Rei

As both KAoS and Rei are policy languages, they both have elements to define different types of policies over different resources. Figure 4.4 shows the basic policy metamodel for Rei and Figure 4.5 shows the basic policy metamodel for KAoS.

However, a close look at these two metamodels shows that the element <policy> does not refer to the same concept in both languages. Indeed, the policy element in KAoS carries a meaning equal to the meaning of the deontic element in Rei (see Figure 4.7). As shown in Figure 4.4, a Rei policy beside all the other elements that might have, defines the <grants> element over a set of GrantingOrDeontic elements. A GrantingOrDeontic element itself is either a Granting element, which consists of a constraint, a DeonticObject (equal to a policy rule), and the Entity to which a deontic object is granted; or it is just a DeonticObject (see Figure 4.6).

Comparing Figures 4.5 and 4.7, it can be easily perceived that these two constructs of the two languages are a lot closer in terms of the meaning they carry. A KAoS policy can have a control action, a triggering action, an oblige action, and a series of conditions. The triggering action, and the set of conditions can be considered equal to the constraint
CHAPTER 4. THE POLICY INTERCHANGE FRAMEWORK

Part of a Rei deontic rule, the controls action can be considered equivalent to the Action in a Rei deontic object, and it will be shown in Figure 4.15 that KAoS embeds the agent as an attribute in the class Action. Consequently, the policy element in Rei does not represent only one single policy rule, but it represents a set of deontic elements, as the relation between the Policy element and the grants sub-element is one-to-many. In KAoS however, each policy element stands for exactly one policy rule. As a result, a policy element in Rei is mapped to a series of policies in KAoS.

As shown in Figure 4.4, Rei defines constraints that can be placed over a policy set to control its behavior. It also defines meta-policies such as Behavior, ModalityPrecedence, MetaMetaPolicy, and Priority to control the behavior of each single deontic rule from the level of a policy set, i.e. the <policy> element (see Figure 4.8). In contrast, KAoS only defines a nonNegativeInteger to put precedence over different policies.
CHAPTER 4. THE POLICY INTERCHANGE FRAMEWORK

Figure 4.4: The policy meta-model for Rei

Figure 4.5: The policy meta-model for KAoS

Figure 4.6: The Granting element in Rei
4.2.3 Constraints and Conditions in Policy Languages

As we mentioned in the previous section, the constraints of Rei (Figure 4.9) can be considered identical to the conditions of KAoS (Figure 4.10a). The core of Rei constraints is the SimpleConstraint which, similar to RDF, uses a subject, an object, and a predicate to represent the relations between two resources. However, as Rei resembles an XML representation of Prolog, boolean operators have been defined as BooleanConstraints to make the definition of conjunctions and disjunctions possible.

In KAoS however, the conditions are a lot more complicated. A condition in KAoS represents a situation based on the current State and the history of the events. As Figure
4.10b shows, the EventHistory keeps the time-stamped events that are mostly in the form of occurred actions, represented in the ActionHistory class. To each ActionHistory, relates a hasRegisteredAction which represents the current action occurred in each time interval, the actor of the action, and the context to which the action has been applied.
4.2.4 Actions in KAoS and Rei

Actions in Rei are classified into two categories, a) *Simple Actions* and b) *CompositeOrSequence Actions*. A simple action (Figure 4.12) is itself a super concept for a domain action. A domain action in Rei can have an actor, a location and a time, a precondition, and an effect. A CompositeOrSequence action is used to define either a composite action, or an order over the execution of the actions. Again, in order to be compliant with RDF, each CompositeAction in Rei can combine two actions at a time, tagged with first and second. The set of actions for a composite action can be tagged as Combination, Sequence, or Choice, identifying different behaviors for the reasoner to deal with a composite action.

KAoS defines a wide range of different actions (Figure 4.14) with special functionalities. An action in KAoS, as shown in Figure 4.15, has a context to which the action is applied, which either is a dataContext of any type (i.e. of type OWLTHING), or an objectContext of
type Target, which is itself an entity. A KAoS action also has one and exactly one Actor, referred to by the performedBy property in the definition of the Action.

Providing mappings from a KAoS action to a Rei action is easy, mainly because the Rei action is a more general concept with an abstract meaning. Upon recognizing the actor and the context of a KAoS action, it can be defined as a Rei action even without any pre-condition, location, time, or effect. However, for a Rei action to be convertible to a KAoS action, the type of the action should be determined. This task is generally not an easy task to be done by machines. However, as both KAoS and Rei are based on OWL, and one of the prominent features of OWL is its expandability, the definitions for actions in KAoS can be reused in Rei, but when it comes to using the Rei reasoner, problems rise as the Rei
reasoner is not able to understand the meaning of the actions in KAoS.

Among all the actions defined in KAoS (Figure 4.14), we refer to two of the most important ones, namely PolicyAction and ObligateAction. PolicyAction is the main action to manage the behavior of the defined policies. For each policy, referred to as involvedPolicy, the Target and the Action can be managed, and their attributes and the corresponding values can be manipulated. The PolicyAction in KAoS is extended to cover a wide range of possible actions over the defined policies, including PolicyModificationAction, PolicyDecisionAction, and PolicyQueryAction (see Figure 4.17). It should be noticed that, not all the PolicyActions will be used to define the policies. Some of the PolicyActions and other elements of KAoS are used for the purpose of reasoning; yet the policies can be placed over any of the PolicyActions to control the behavior of these actions.

![Figure 4.16: The meta-model for KAoS PolicyAction](image)

![Figure 4.17: The meta-model for KAoS PolicyAction-Types](image)
ObligateAction is another important action in KAoS. An ObligateAction makes the active actor to execute the ObligateAction once the triggering action happens and the execution constraints for the obligation action are met (see Figure 4.18).

![Figure 4.18: The meta-model for KAoS ObligateAction](image)

### 4.2.5 Actors in KAoS and Agents in Rei

In Rei, any Entity can be declared as an agent. The agent is an individual or a group of individuals to whom the policy is applied. However, Rei doesn't place any constraint over the characteristics of the agents. As Figure 4.19b shows, the class Agent is inherited from the class Entity which is itself a simple OWL class with two properties of owner and affiliation, the first one referring to the owner of an entity, and the second one referring to the resource with which the entity is affiliated (Figure 4.19a).

![Figure 4.19: The meta-model for a) Rei Entity and b) Rei Agent](image)

In KAoS, again a detailed hierarchy of entities and actors have been defined. In KAoS, the concept for entities has been shaped a lot better than in Rei. Rei has loose definitions for entities (cf. Figure 4.19a) and the range or the domain for many of the concepts has been left undefined, referring to OWL's concept of THING which is identical to any resource of any type. Conversely, according to Figure 4.20 for KAoS, the resource is considered equal to the entity and various concepts have been inherited from the general concept of Entity.

Actors in KAoS, represent a complete set of possible agents that might interact with the system (Figure 4.21). Different physical and artificial actors have been defined by the
system which helps with precise classification of the actors while working with policies. The only problem with the obtained model, as we have highlighted in Figure 4.21, is the redundant definition of both Human and Person which seem to be identical. The reason behind distinguishing between these two concepts by the developers of KAoS is not clear to the author of this dissertation. Furthermore, according to the ontology definition of KAoS for the concept Human, it inherits from both Person and PhysicalActor. Since Person itself has been already defined as a subclass of PhysicalActor, the inheritance from PhysicalActor by Human seems to be redundant. Although Rei refers to all the actors of the policy as agents, Figure 4.21 shows that KAoS considers agent a SoftwareActor, and it is not the only type of actor that the system can deal with.

An actor in KAoS can be controlled by a set of policies, and is capable of performing a series of actions. It also can cooperate with other actors as a team (Figure 4.22). While mapping the actors from KAoS to Rei is pretty simple because of going from a specific definition to an abstract definition, the mapping of actors from Rei to KAoS is hard if an exact mapping to an appropriate type for the actor is intended.
4.2.6 Types of Policies in KAoS and Rei

Now that we have recognized the basic elements of both languages, we need to realize what types of policies are covered by either of the languages. The types of the policies that these languages can cover are in a direct relationship with the functionalities that they offer.

Figure 4.23 represents the types of policies that are covered in KAoS. As the figure shows, a policy in KAoS is either an AuthorizationPolicy or an ObligationPolicy with each of them further specialized to PosAuthorizationPolicy, NegAuthorizationPolicy, PosObligationPolicy, and NegObligationPolicy. It can also be a policy over the policies that conflict, ConflictedPolicies, which usually is not considered as a policy type, but it is only a subclass of the class Policy. An AuthorizationPolicy introduces the set of permissions and prohibitions for an actor while dealing with a context. However, as we mentioned in Section 2.4, KAoS and Rei are context-based policy languages that define permissions and prohibitions not over the roles of the actors but over the context of interactions. Thus, rather than the actor of an action, it is the action itself that is denied or allowed. For an actor to be able to do an action, it must be capable of satisfying another
actions which describe the constraints.

Figure 4.23: The meta-model for K AoS policy types

Figure 4.24 is an illustration of different types of policies supported by Rei. Interestingly, Rei also has the four main types of policies that have been defined in K AoS. As we mentioned in Section 3.3.1, *Permission*, *Prohibition*, *Obligation*, and *Dispensation* in Rei are respectively equivalent to *PosAuthorizationPolicy*, *NegAuthorizationPolicy*, *PosObligationPolicy*, and *NegObligationPolicy* in K AoS.

Nevertheless, the *obligation* policy has a more complicated definition than the definition that Rei offers for the obligation policy. Figure 4.25a shows the definition of the obligation policy for K AoS and Figure 4.25b illustrates the definition for the element *ObligationConstraint*. Finally, Figure 4.26 represents different subclasses for an *ObligationConstraint*. Upon satisfaction of an *ObligationConstraint*, the obligated policy is applied and the corresponding action is enforced.
Figure 4.25: The meta-models for (a) an ObligationPolicy and (b) an ObligationConstraint in KAoS

Figure 4.26: The meta-model for KAoS ObligationConstraint types
4.2.7 SpeechActs in Rei

So far, we have mostly discussed the elements that are similar between the two policy languages. However, there are several nonconformities between the two languages as well. **SpeechActs** in Rei are primarily used for dynamic and remote policy management [88]. Figure 4.27a illustrates the meta-model for different types of SpeechActs in Rei and Figure 4.27b shows the constituents of a Rei SpeechAct element. It is worth-noting that KAoS does not have any feature to cover the concepts of SpeechActs. As a result, no mapping for these elements to KAoS is possible.

![Figure 4.27: The meta-model for a) Rei SpeechAct Types and b) Rei SpeechAct element](image)

4.2.8 The concepts specific to KAoS

There are concepts in KAoS which are not available in Rei. Most importantly, KAoS represents groups and teams to classify the actors and the domains of interaction. Figure 4.28 shows different groups that KAoS defines. Figure 4.29 illustrates the definition of a Group element. Figures 4.30 and 4.31 show the concepts for Domain and Team as the subconcepts of the concept Group, for KAoS.

4.3 Transforming the Policies to/from R2ML

Given the full representation of KAoS, Rei, and R2ML (both syntactically and semantically), we can take the main step in defining the transformations between these three languages, aiming at providing an exchange method between different concepts of these languages. It should be noted that although we deal with the syntactical mapping of the elements, the
CHAPTER 4. THE POLICY INTERCHANGE FRAMEWORK

Figure 4.28: The meta-model for KAoS Group Types

Figure 4.29: The meta-model for the Group element in KAoS

Figure 4.30: The meta-model for KAoS Domain concept
process of mapping considers semantic similarities as well. In this section, we show that, despite having several possibilities to map the elements of one language to the elements of another language, we choose those elements which are most similar semantically. Our transformations in this section almost follow the QVT’s graphical notations [85] for transforming between concepts. However, we have included some dashed arrow lines to make the one-to-one mappings easier to understand.

In our introduction to R2ML, in Section 3.2.7 we presented the major types of rules that R2ML covers. Among all the rules, derivation rules seem to be better options to model the policies, mostly because they entail the meaning of: “obtaining a conclusion upon satisfaction of a set of conditions (or credentials)”’. Using implication in integrity rules makes them quite close to our intentions too, but as we already mentioned, integrity rules are generally used where the set of conditions of the rule are persistent over time. Reaction rules can be used to reflect the pre- and post-conditions of a policy rule, especially while communicating with Web services. Nonetheless, we believe the intention in using policies is closer to the meanings expressed by derivation rules, i.e. deriving the conclusions once the conditions are met. Thereupon, we opt the use of derivation rules in order to model policy languages. Some other research works covering this same area are available in [51, 52, 53]

In our representation of different types of policies for KAoS and Rei (see Figure 4.23 and 4.24 respectively), we showed that there are generally four main types of policy rules
that these languages support. PosAuthorizationPolicy, NegAuthorizationPolicy, PosObliga-
tionPolicy, and NegObligationPolicy in KAoS are equivalent to Permission, Prohibition, 
Obligation, and Dispensation in Rei. To be able to model these concepts in R2ML, and to 
be compliant with both languages, we decided to define some general classes for these 
types of policies using R2ML. Due to the simplicity of the namings for the policies in Rei, 
we opted the same set of names to define our policy classes in R2ML. We also mentioned 
that R2ML has its own vocabulary, so it is possible to exploit R2ML vocabulary to define 
these policy concepts. Figures 4.32a and 4.32b show the metamodels for the class Policy 
and its subclasses that we have defined in R2ML. As it can be seen in the models, these 
classes have simple definitions at the moment which currently seem satisfactory for our 
purpose. However the metamodels for these concepts will be developed in the future, aiming 
at providing a General Policy Modeling Language by using R2ML. The definition of these 
concepts in R2ML Vocabulary can be found in Program 4.2. Again it should be noted that 
the properties of these general R2ML-Policy classes do not have any range and they refer 
to any available resource. This is mainly to increase the flexibility of our R2ML classes to 
work with different actions and policies from different policy languages. We may decide to 
make them more restricted in the future.

While transforming each single policy rule from either KAoS or Rei to R2ML, we decide 
about how to map these policies to R2ML policies by choosing the same naming conventions 
that we introduced at the beginning of this section. Next we describe how the concepts of 
R2ML along with the classes that we defined above, are jointly used to provide a meaningful 
mapping between the concepts of the two policy languages.

4.3.1 Transforming Policies from KAoS to R2ML

In our syntactical descriptions of KAoS and Rei given in Section 3.3, we have mentioned 
that a KAoS policy is an object of the Policy class in KAoS Policy Ontology (KPO) [99] 
with its attributes instantiated to a set of users, events, and resources that make the policy 
fire. Considering the KAoS policy element (e.g. Programs 2.4 and 2.5) as a rule, the control 
element is executed upon the occurrence of the events described in the requiresConditions 
element (Figure 4.5). The control element in KAoS refers to an action. It can itself place 
a series of constraints on the definition of the action that can be executed by the policy. 
The main advantage in using KAoS is its flexibility of expansion. The Action class in KAoS
CHAPTER 4. THE POLICY INTERCHANGE FRAMEWORK

Program 4.2: The R2ML Vocabulary for our defined R2ML Policy Model
can be easily expanded thanks to the use of OWL, to fully capture the meaning of the final action that is desired to be executed. Placing constraints on different properties of this action can also specialize the meaning that is desired to be transformed.

Once a KAoS policy rule is fired, the decision over whether or not to perform the action of the control element is made, an obligation of execution is placed over the obligation action, and the effect of the action is enforced to the current state of the system. This means, to model a KAoS policy with a derivation rule, we need to place the content of the controls element, the oblige element, and the effect element (if there is any) in the conclusion part of the derivation rule. Whatever else that can lead to making such a decision should be placed in the condition part of a derivation rule. This includes the content of the requiresConditions element, the triggerAction, and also the variables that initialize the values for the elements in the conclusion of a rule (see Section 3.2.7 and Figure 4.33). Figure 4.33 shows the possibility of transforming a KAoS policy rule to a derivation rule in its most abstract model. The main issue is to decide how and using what R2ML elements this process of transformation can be best achieved. The conclusion of a derivation rule should be only one R2ML Atom, now the question is how to manage all the actions in one single Atom.

The first step is to convert the KAoS Policy class to our R2ML policy class. An instantiation of the derived R2ML policy class will be placed as a conclusion in the derivation rule. Figure 4.34 shows how this mapping can happen.
CHAPTER 4. THE POLICY INTERCHANGE FRAMEWORK

Figure 4.33: The abstract transformation of a KAoS Policy to a R2ML derivation rule

Figure 4.34: The abstract model of mapping a KAoS Policy class to our R2ML policy model
As we argued in Section 3.2.4, a KAoS ObjectDescriptionAtom describes a set of properties and attributes, referred to as slots, for an object called the subject. As for the policy rules, the conclusion of a policy rule is a policy element (or object) with the values for the appropriate action to be taken, the actor to which giving the permission is allowed or denied, the context to which the policy is applied, and the priority of the policy rule. As a result, an ObjectDescriptionAtom, with a policy object as its subject is the best element to model the conclusion of a derivation rule. Figure 4.35 shows how we instantiate a policy object from our R2ML Policy class. It illustrates how the policy object is mapped to the subject of the R2ML ObjectDescriptionAtom, the corresponding class for this atom is set to the Policy class and different attributes and properties for a policy object are mapped to corresponding ObjectTerms or DataTerms depending on whether the attribute is a DataAttribute or an ObjectAttribute. It has also been shown in Figure 4.35 that an ObjectDescriptionAtom can have several number of slots to cover various number of data or object attributes for a policy. It makes it the appropriate R2ML element, both syntactically and semantically, to be used for the purpose of transforming between the objects of the policy rules.

However, referring to Figure 4.23, one should note that the class Policy is an abstract class, and so is our R2ML Policy definition of Figure 4.32. Thus in Figure 4.35, the class Policy is usually replaced with one of its concrete sub-classes, i.e. Permission, Prohibition, Obligation, or Dispensation.
Having the conclusion of our derivation rules constructed, we need to extract its condition part from K AoS policy rules as well. The condition part of a policy derivation rule contains the definition of the variables, actions, actors, conditions, and any other construct that result in deriving the final policy object. Although the conditions of a policy rule could be modeled with \textit{ObjectDescriptionAtoms}, we chose \textit{ReferencePropertyAtoms} mainly to be compliant with the definitions of Rei (defined in the form of triples - see Section 2.4.3) and also other R2ML transformations (e.g. transformations between F-Logic and R2ML also have \textit{ReferencePropertyAtom} in the condition part). It simplifies the later conversions of the policies to other rule languages for which we have R2ML transformations already defined. On the other hand, a \textit{ReferencePropertyAtom} triple models a binary predicate. A set of \textit{ReferencePropertyAtoms} with the same subject element can always be combined and converted to any element of higher arity (e.g. \textit{ObjectDescriptionAtom}), and thus using \textit{ReferencePropertyAtom} does not contradict the use of \textit{ObjectDescriptionAtom}. Furthermore, in our case, \textit{ReferencePropertyAtoms} carry even a better semantic meaning for the transformations. Semantically they are equivalent to an OWL object property, and as K AoS is nothing but pure OWL, they model object properties of K AoS too. Figure 4.36 shows the conversion of a K AoS property to a ReferencePropertyAtom in R2ML. The converted element is then placed in the \textit{conditions} part of a derivation rule.

![Diagram](image)

**Figure 4.36**: The abstract transformation of a K AoS property to a R2ML ReferencePropertyAtom

In some cases it happens that all the objects are not named-objects, but rather, they refer to ObjectVariables. In K AoS, due to the impossibility of representing variables in OWL,
a *role-value-map* method is used in which a class representing the set of possible elements for a slot replaces the object variable in order to show all the possible options an element can take. The transformation procedure should recognize these classes and convert them to appropriate R2ML elements. This happens in two steps during our mappings. Once the role-value-mapped class is determined, it is first mapped to an *ObjectClassificationAtom* with a variable name assigned to it, and then the variable is used in place of the required elements for the class, for example in a *ReferencePropertyAtom*. Figure 4.37 shows the conversion of a role-value-map class to a R2ML *ObjectClassificationAtom*. In this Figure, the generated variable x shows the variable that later can be used in other places in the rule.

![Figure 4.37: The abstract transformation of a KAoS role-value property to a R2ML ObjectClassificationAtom](image)

For example in case we need to convert it to the *ReferencePropertyAtom* of Figure 4.36, the only change would be to replace actor1 with x. It should be highly noted that, an *ObjectClassificationAtom*, representing a variable, always needs to be placed in the condition part of the derivation rule, even if the variable needs to be used in the conclusion part. Figure 4.38 also shows another example of how the combination of *ObjectClassificationAtom* and *ReferencePropertyAtom* can be used to define a class of actors in KAoS.

A KAoS policy might also have a *trigger element*. To the best of our knowledge, this element is only used with *NegObligation*- and *PosObligation-Policies* showing a set of events that trigger the occurrence of an action. In our *R2ML Policy* model, we have considered a slot for these actions. Furthermore, the detailed information about these actions is defined as *ReferencePropertyAtoms* in the condition part of a derivation rule. Consequently, the
same process shown in 4.36 is also applied to the trigger element, which itself refers to an action. The *obligeAction* element of a KAoS policy is also considered as an ObjectSlot in the ObjectDescriptionAtom of the R2ML Policy object. The ObjectSlot refers to the action that should be performed by the actor as an obligation, once the policy rule is fired.

Having the most important elements of a KAoS policy explained here, we present some of the concrete, syntax-based formalization of our transformation rules in Table 4.1. The XSLT implementations of our transformations are available in [117].

Other elements of the KAoS policy language are placed in the body of a derivation rule. As a result we perform a conversion to ReferencePropertyAtom for those elements. The main reason is to be compliant with Rei. Moreover, once we have the relations defined in the form of RDF triples, then conversion to the predicates of higher arity is a lot simpler,
### Table 4.1: The concrete XML-based conversion of some of the KAoS elements to R2ML elements

<table>
<thead>
<tr>
<th>KAoS Element</th>
<th>R2ML Element</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;policy:PosAuthorizationPolicy</code></td>
<td><code>&lt;owl:Class rdf:ID = &quot;Policy_CommunicationCertificate&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;policy:requiredConditions rdf:resource = &quot;#Policy_TrustedEntity&quot;&gt;</code></td>
<td><code>&lt;owl:Restriction&gt;</code></td>
</tr>
<tr>
<td><code>&lt;policy:controls</code></td>
<td><code>&lt;owl:Restriction&gt;</code></td>
</tr>
<tr>
<td><code>rdf:resource = &quot;#Policy_CommunicationCertificate_Action&quot;/&gt;</code></td>
<td><code>&lt;/owl:Class&gt;</code></td>
</tr>
<tr>
<td><code>&lt;policy:PosAuthorizationPolicy&gt;</code></td>
<td><code>&lt;owl:Class&gt;</code></td>
</tr>
<tr>
<td><code>&lt;r2ml:DerivationRule&gt;</code></td>
<td><code>&lt;owl:Class&gt;</code></td>
</tr>
<tr>
<td><code>&lt;r2ml:conditions&gt;</code></td>
<td><code>&lt;owl:Restriction&gt;</code></td>
</tr>
<tr>
<td><code>&lt;owl:Class rdf:ID = &quot;Policy_CommunicationCertificate_Action&quot;&gt;</code></td>
<td><code>&lt;owl:requiredProperty rdf:resource = &quot;http://ontology.ihmc.us/Action.owl#performedBy&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;owl:allValuesFrom</code></td>
<td><code>&lt;/owl:Class&gt;</code></td>
</tr>
<tr>
<td><code>&lt;owl:Class&gt;</code></td>
<td><code>&lt;owl:Restriction&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/owl:Class&gt;</code></td>
<td><code>&lt;/owl:Class&gt;</code></td>
</tr>
<tr>
<td><code>&lt;owl:Restriction&gt;</code></td>
<td><code>&lt;owl:Restriction&gt;</code></td>
</tr>
<tr>
<td><code>&lt;owl:allValuesFrom</code></td>
<td><code>&lt;owl:allValuesFrom</code></td>
</tr>
<tr>
<td><code>&lt;owl:referenceProperty rdfs:domain = &quot;http://ontology.ihmc.us/Policy.owl#hasPartner&quot;&gt;</code></td>
<td><code>&lt;owl:allValuesFrom</code></td>
</tr>
<tr>
<td><code>&lt;/owl:Class&gt;</code></td>
<td><code>&lt;owl:Class&gt;</code></td>
</tr>
</tbody>
</table>

The mappings for the condition part go here: `<!...>`

The mappings for the conclusion part go here: `<!...>`

The mappings for the conclusion part go here: `<!...>`

The mappings for the conclusion part go here: `<!...>`

The mappings for the conclusion part go here: `<!...>`

The set of constraints on the action in KAoS are defined as intersection of a series of restrictions in OWL: `<!...>`

We can use either `r2ml:ObjectName` or `r2ml:ObjectName` to define the actors: `<!...>`

The set of constraints on the action in KAoS are defined as intersection of a series of restrictions in OWL: `<!...>`

The set of constraints on the action in KAoS are defined as intersection of a series of restrictions in OWL: `<!...>`
as for n triples with similar subject we can make one ObjectDescriptionAtom of arity n by combining all these elements.

4.3.2 Transforming Policies from Rei to R2ML

A Rei policy, similar to KAoS, is an instantiation of its Policy class, defined in the Rei ontology [88]. However, a policy element in Rei represents a list of rules (each defined as a deontic child element) while a policy construct in KAoS represents only one rule. Each R2ML derivation rule is also equivalent to one policy rule. Therefore, converting a policy from Rei to KAoS or to R2ML may result in having more than one KAoS policy or R2ML rule. Furthermore, as Rei deals with variables similar to the way Prolog treats them, and because variables can have different values during the run-time, a single policy in Rei might be converted to more than one KAoS policy or based on different combinations of values that the variables can take. Fortunately R2ML can accept a set of derivation rules too. This way a Rei policy element is transformed to a R2ML derivation rule set, in case more than one policy rule can be derived from a Rei policy.

Our R2ML rule structure for the transformations is more similar to the structure of the Rei language than to KAoS and hence it is easier to convert a Rei policy rule to R2ML. First we start with the top most level which is converting a Rei policy rule to our R2ML policy model. However, one single Rei policy rule is a combination of the elements in its definition of the policy set as well as the elements in a deontic rule and thus a derivation rule should employ the elements from both of these Rei elements. Figure 4.39 shows how this conversion from a Rei policy to a R2ML derivation rule can happen.

As Figure 4.39 shows, a derivation rule is formed by placing the Rei constraints in the condition part. All the boxes in red are constraints that can be placed in the condition of a derivation rule. On the other hand, the concepts that go to the conclusion should be embedded as slots in the R2ML policy that we defined for our models (cf. Figure 4.32 and Program 4.2). The next step would be to define the appropriate place for these elements in the ObjectDescriptionAtom that specializes our R2ML class for a Rei policy. Figure 4.40 shows how the mapping can be done from a Rei Policy object to an ObjectDescriptionAtom for an R2ML Policy class. The figure clearly shows that the information about the context and the action are mapped to our policy, but the actor of an action relates to its
corresponding action using a ReferencePropertyAtom which is placed in the condition part of the derivation rule. Our current R2ML Policy Model does not support the concepts that are shown in blue in Figure 4.40. It is not hard to extend the R2ML Policy Model to cover these concepts, but we believe that the definition of most of these concepts is beyond the intentions of a single policy rule. More research should be conducted to identify the correct place for these elements during the process of transformation.

Rei uses SimpleConstraints and BinaryConstraints to define the conditions of a deontic rule. All constraints for a deontic element are considered as preconditions of that deontic rule and treated the same way as requiresConditions element in KAoS. Figure 4.41 shows how the SimpleConstraint in Rei is mapped to R2ML’s ReferencePropertyAtom.

As shown in the Figure, the conversion is one-to-one between the corresponding elements of Rei’s SimpleConstraint and R2ML’s ReferencePropertyAtom, which should be enforced during the process of mapping. The ReferencePropertyAtom completely suits our needs to express the SimpleConstraints of Rei. Similarly, the binary constraints can be neatly covered while using logical formulas. Recall that the logical formula is used in both the conclusion and the condition part of a derivation rule. Figure 4.42 shows the class diagram
Figure 4.40: The abstract meta-model for converting between the element of Rei policy and R2ML policy.
of these two concepts and how the correspondence between their semantics occurs.

As it has been highlighted in Figure 4.42, a logical formula supports both \textit{StrongNegation} and \textit{NegationAsFailure}, the first one as a concept in Description Logic, and the second one a concept in Computational Logic (cf. Section 3.3.2). This shows the flexibility of R2ML in covering the concepts of either worlds. Yet, care should be given while transforming the R2ML rule to a language with a certain type of negation, e.g. Rei as a language based on Computational Logic supports only \textit{NegationAsFailure}, and not \textit{StrongNegation}.

As mentioned earlier, a deontic element in Rei has an action and an actor as its child
elements. The actor is related to the action defined in the R2ML Policy object through a performedBy property under the R2ML ReferencePropertyAtom and is placed in the condition part of a rule, similar to what we have done for KAoS. The actions in Rei are defined either by using Rei elements or OWL classes. For mapping the actions to R2ML we can just refer to the already defined action or use R2ML vocabulary to redefine it. A policy element in Rei has also a child element, named context. In our R2ML transformation, we copy this same context for all our derivation rules and place it as an ObjectSlot into our R2ML Policy ObjectDescriptionAtom.

Thanks to the use of ReferencePropertyAtom construct in R2ML, we can easily convert each tripled constraint of Rei to R2ML through a one-to-one mapping of the subject, the object, and the predicate elements.

A Rei action can be described also as an ObjectDescriptionAtom in R2ML. It is required in some cases where we need to carry the whole meaning of an action for a defined object. In such situations the ObjectDescriptionAtom is placed in the condition part of the derivation rule with its subject used by the other elements either in the condition or in the conclusion of the rule. The subject of an ObjectDescriptionAtom in this case can be replaced by a variable. Figure 4.43 illustrates how the action in Rei can be converted to an ObjectDescriptionAtom in R2ML.

Table 4.2 shows an example of the concrete XML-based mapping of two of the Rei elements to R2ML, namely the deontic element to a derivation rule and the SimpleConstraint to a ReferencePropertyAtom.

4.3.3 Transforming Policies from R2ML to KAoS/Rei

Transforming the R2ML derivation rules back to either of these policy languages is a much simpler task now that we have the transformation rules defined. Of course, the mappings are bidirectional. That is, the same way that, for example, a SimpleConstraint element of the Rei language would be converted to a ReferencePropertyAtom in R2ML and placed in the condition part of a derivation rule, a ReferencePropertyAtom in the condition part of a rule can be converted to a SimpleConstraint in Rei. However, there are some concepts that are not simple to map. For example one may think how an OWL class will be created.
out of a set of ReferencePropertyAtoms. As we have already mentioned, the set of ReferencePropertyAtoms with the similar subject will create an OWL class with the properties modeled as restrictions on values in the object part of the ReferencePropertyAtom. To make an OWL class out of a set of ReferencePropertyAtoms we combine all the atoms that have the same subject. The other properties are defined as the restrictions over the class to which the subject element belongs. As an example, Figure 4.44 shows how a KAoS action, as an OWL class, can be made out of various ReferencePropertyAtoms.

Mapping of a R2ML ObjectDescriptionAtom to a SimpleConstraint in Rei is almost similar to the process of converting a KAoS class to a R2ML ReferencePropertyAtom, as in both cases we are going from an element of higher arity to an element of lower arity. In particular, a ReferencePropertyAtom in R2ML as we showed in Figure 4.41 is almost equivalent to a SimpleConstraint in Rei. Figure 4.45 shows how the ObjectDescriptionAtom in R2ML can be converted to the SimpleConstraint in Rei.

Another problem in our transformations happens when dealing with priorities. Priorities in KAoS are defined with numbered values but in Rei we have meta-rules to give priority to one rule over the other. So the priorities in Rei are a set of rules placed over policy rules.
Table 4.2: The concrete mapping of two of the Rei elements to their corresponding R2ML elements

<table>
<thead>
<tr>
<th>Rei Element</th>
<th>R2ML Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;deontic:Permission rdf:ID=&quot;PolicyName&quot;&gt;</td>
<td><a href="">r2ml:DerivationRule</a></td>
</tr>
<tr>
<td>&lt;deontic:action rdf:resource=&quot;#actionToPerform&quot;/&gt;</td>
<td><a href="">r2ml:conditions</a></td>
</tr>
<tr>
<td>&lt;deontic:actor rdf:resource=&quot;#ActorSet&quot;&gt;</td>
<td><a href="">r2ml:condition</a></td>
</tr>
<tr>
<td>&lt;deontic:constraint rdf:resource=&quot;#ConstraintsSet&quot;/&gt;</td>
<td><a href="">r2ml:conclusion</a></td>
</tr>
<tr>
<td>&lt;/deontic:Permission&gt;</td>
<td></td>
</tr>
</tbody>
</table>

As Figure 4.4 shows, a policy in Rei can have a rule priority element in which it states whether a deontic rule has a greater or lesser priority as compared to another rule. Thus it is hard to convert the priorities for a series of rules in Rei to numbered priorities that are compliant to KAoS. We have tried to solve this problem by converting the meta-rules of Rei to a numbered model in our R2ML transformation and then during conversion from R2ML to Rei we convert the numbers to a form of meta-rules compliant with the syntax of Rei. However, further research in such areas should be conducted.

4.4 Summary

This chapter was a detailed discussion of the possibilities for transforming between different policy languages knowing their syntactical and semantic meanings. We showed that the mappings do not happen only at the syntactical level of the languages, but the semantics are also carried while providing mappings. The semantics of the conversions are partly covered when we bridge between the logic of the policy languages. The other part is obtained while having possible options for R2ML elements, we chose the ones that best describe our intentions. For instance, we showed that although some elements of either of the policy languages can be modeled by both ReferencePropertyAtoms and ObjectDescriptionAtoms, we
chose between them depending on the situation and the place they are going to be used at. \textit{ObjectDescriptionAtoms} are used to map the concepts that have a more descriptive content, like the ones to define the policies, whereas we chose to use \textit{ReferencePropertyAtoms} when there is a need to emphasis on the predicate which is evaluated as a credential for the policy rule.

In Section 4.2, we deeply described the policy languages by presenting their meta-models, which is probably the most intuitive way of understanding the underlying concepts of the languages. We compared the similarities and dissimilarities of all different policy languages and showed how and where the mapping between the concepts of different languages can happen. In Section 4.3 we added R2ML as the additional policy language to be used for the purpose of transformations. Furthermore, The possibility of converting between different
elements of R2ML, Rei, and KAoS was shown. In the next chapter we provide some concrete examples of how the mappings can be performed between these two policy languages. In Chapter 6 we also discuss some of the main problems we foresee for this framework.
Chapter 5

Examples of Transforming between Policy Rules

To this point, we have conceptualized and formalized our policy transformation framework. Now that the formalization of our transformations is complete, we can apply our definitions to some real samples of either of the two policy languages, i.e. KAoS and Rei. Employing our abstract mapping models, we try to justify the transformations. Section 5.1 takes some KAoS examples and converts them to R2ML by applying our transformations of the previous chapter. Section 5.2 starts with Rei policies and describes their equivalents in R2ML. Sections 5.3 and 5.4 convert the R2ML derivation rules of Sections 5.1 and 5.2 to Rei and KAoS respectively.

We should point out that there are some concepts in both languages which are not convertible to the concepts of the other policy language, that is, there is no equivalent element in the other policy language to cover those concepts. In our examples of this chapter we avoid using those elements in the sample policies. We mentioned some of the differences between the concepts of Rei and KAoS in Sections 4.2.7 and 4.2.8, and in Chapter 6 we explore the complete description of the elements of each policy language which are not fully covered by the other language.
5.1 KAoS Policies to R2ML Derivation Rules

5.1.1 Printer Access Policy by Sky Team Customers

As the first example, we start with Programs 2.4 and 2.5. Program 2.4 shows a KAoS rule which controls a PrinterAccessAction. The policy is a PosAuthorizationPolicy and instantiates hasSiteOfEnforcement and hasPriority. The PrinterAccessAction is performed by a class called SkyTeamCustomer which has been described in Program 2.5.

As we stated in the previous chapter, we define four main policy classes for our policy rules, i.e. Permission, Prohibition, Obligation, and Dispensation (cf. Section 4.3). The first step in providing the mappings is to determine the appropriate R2ML classes and elements that can be used according to our definitions in the previous chapter. A PosAuthorizationPolicy in KAoS is equivalent to our definition of the Permission class in R2ML. Having a KAoS policy element necessitates the definition of an ObjectDescriptionAtom to describe its properties, which should be placed in the conclusion part of a derivation rule. The ObjectDescriptionAtom takes the name of the policy instance as its subject and is considered as an instance of the Permission class that we defined in the previous chapter. hasSiteOfEnforcement from a KAoS policy is modeled as an ObjectSlot for our ObjectDescriptionAtom with its referencePropertyID referring to context according to our definition of R2ML Policy class. The hasPriority attribute in KAoS has an equivalent hasPriority element in our R2ML Policy model which is considered as a DataSlot.

In our example, we have a PrinterAccessAction as a subclass of AccessAction for which we have the metamodel illustrated in Figure 5.1. As the figure shows, accessedEntity
is the subProperty of hasObjectContext and thus the class that this property refers to (i.e. Printer31-57 in our example) plays as the context for the PrinterAccessAction. Summing up, the obtained ObjectDescriptionAtom for this policy element can be expressed as in Program 5.1.

```xml
<r2ml:ObjectDescriptionAtom r2ml:classID="r2ml_policy:Permission"
<r2ml:subject>
<r2ml:ObjectName r2ml:objectID="SkyTeamGate31-57PrinterAccess"/>
</r2ml:subject>
<r2ml:ObjectSlot r2ml:referencePropertyID="r2ml_policy:hasContext">
<r2ml:object>
<r2ml:ObjectName r2ml:objectID="&some-ontology;TargetSite"
 r2ml:classID="kaos_policy:SiteOfEnforcement"/>
</r2ml:object>
</r2ml:ObjectSlot>
<r2ml:DataSlot r2ml:attributeID="r2ml_policy:hasPriority">
<r2ml:value>
<r2ml:TypedLiteral r2ml:lexicalValue="10"
 r2ml:datatypeID="xs:int"/>
</r2ml:value>
</r2ml:DataSlot>
<r2ml:ObjectSlot r2ml:referencePropertyID="r2ml_policy:hasAction">
<r2ml:object>
<r2ml:ObjectVariable r2ml:name="X"/>
</r2ml:object>
</r2ml:ObjectSlot>
</r2ml:ObjectDescriptionAtom>
```

Program 5.1: The R2ML ObjectDescriptionAtom excerpt of the KAoS Policy Program 2.4

Having a look at the SkyTeamGate31-57PrinterAccessAction class, as the to-be-controlled class, we realize that SkyTeamCustomer, Printer31-57 and AccessAction are OWL classes. To be able to place these elements in the conclusion part of a derivation rule, we need to associate the classes with variables which show the extensional meaning of the classes. As Program 5.1 shows, the object for hasAction refers to variable X. ObjectClassificationAtoms seem to be appropriate R2ML constructs to connect a variable to its class name. However they need to be placed in the condition part of the derivation rule. Program 5.2 shows how we can define the relations between the classes and the variables. Program 5.2 also shows the extracted relations between the action, its actor, and its context.
Getting to this point, we have mapped almost everything in Program 2.4 to R2ML, but this KAoS excerpt has another piece of code shown in Program 2.5 which can be placed in our R2ML rule as well. Program 2.5 provides a description for the class SkyTeamCustomer, considering it as a subclass of the class Customer with its firm property only limited to the class SkyTeamAlliance. As it is a descriptive class, it should be placed in the condition part of a derivation rule, and ObjectDescriptionAtom is probably the best element to map this class to. Program 5.3 shows the ObjectDescriptionAtom that we obtain as the result of applying our transformations.

Now we have converted all the elements to the R2ML equivalents. Nevertheless, Program 5.3 needs special clarifications. It can be seen in the program that the classID for our defined ObjectDescriptionAtom is Customer and the classID for its subject element is SkyTeamCustomer. We put this convention in our transformations that the class that is represented for the subject of an ObjectDescriptionAtom, if different from the class for the ObjectDescriptionAtom itself, is always a sub class of it. That is, in our example, the class SkyTeamCustomer is a sub class of Customer, otherwise these two classes should refer to the same name or the class for the subject should be left undefined which would be considered identical to the classID of the object. While transforming these elements to Rei, an ObjectDescriptionAtom in the condition part will be converted back to an OWL class definition which will be referenced in the Rei policy rule. The idea of representing OWL classes as ObjectDescriptionAtoms was mainly to go around representing the concepts of KAoS as a separate vocabulary and to make the whole rule less complicated. However, we will consider the development of this approach to include r2ml vocabulary to our R2ML transformations in the next versions. This will be discussed in more details in Chapter 6.

The final task that we need to do is to place these excerpts in the appropriate places in a derivation rule. Knowing the metamodel for the derivation rule (cf. Figure 3.24), we can simply place Programs 5.2 and 5.3 in the conditions part and Program 5.1 in the conclusion part of the code excerpt of Program 5.4.
PROGRAM 5.2: THE R2ML CLASSIFICATIONATOM EXCERPT OF THE KAoS POLICY PROGRAM 2.4
CHAPTER 5. EXAMPLES OF TRANSFORMING BETWEEN POLICY RULES

Program 5.3: The R2ML ReferencePropertyAtom excerpt of the KAoS Policy Program 2.5

```
<r2ml:ObjectDescriptionAtom r2ml:classID="&some-ontology;Customer">
  <r2ml:subject>
    <r2ml:ObjectVariable r2ml:name="Y"
      r2ml:classID="&some-ontology;SkyTeamCustomer"/>
  </r2ml:subject>
  <r2ml:ObjectSlot r2ml:referencePropertyID="&some-ontology;firm">
    <r2ml:object>
      <r2ml:ObjectVariable r2ml:name="W"
        r2ml:classID="&some-ontology;SkyTeamAlliance"/>
    </r2ml:object>
  </r2ml:ObjectSlot>
</r2ml:ObjectDescriptionAtom>
```

Program 5.4: The R2ML DerivationRule excerpt

```
<r2ml:DerivationRuleSet>
  <r2ml:DerivationRule>
    <r2ml:conditions>
      <!-- THE CONDITIONS GO HERE -->
    </r2ml:conditions>
    <r2ml:conclusion>
      <!-- THE CONCLUSION GOES HERE -->
    </r2ml:conclusion>
  </r2ml:DerivationRule>
</r2ml:DerivationRuleSet>
```

Program 5.4: The R2ML DerivationRule excerpt
5.1.2 Communicating to Service Providers which Support X.509 Certificates

For the next example consider a service provider B which is trying to communicate to different parties. However, the service provider is only willing to communicate to the parties that are already authorized either by the service provider itself or by one of the trusted parties. The corresponding KAoS policy rule for this policy can be found on the KAoS ontology website and has been defined by the developers of KAoS as: "Service Provider B cannot accept data from parties that have not been previously authorized by itself or a trusted party that has been delegated such authorization rights". Program 5.5 gives the representation of this policy in KAoS. It is also available on the KAoS ontology website.

To convert the KAoS rule of Program 5.5, we again apply our conventions from Chapter 4. The rule of Program 5.5 is a NegAuthorizationPolicy rule and thus we describe it using our R2ML Prohibition Class. Consequently, what we need is an ObjectDescriptionAtom with its classID set to Prohibition and its subject named as PolicyLDataAcceptance. Program 5.6 shows how this ObjectDescriptionAtom should look like. This ObjectDescriptionAtom, similar to the one in Program 5.1, is placed in the conclusion part of our policy derivation rule. It should be also regarded that this ObjectDescriptionAtom does not have any hasContext slot because there is no site of enforcement in the original KAoS policy.

As one can see in Program 5.6, X has been defined as a variable for which the corresponding class should be PolicyLDataAcceptance_Action. For PolicyLDataAcceptance_Action, we have the actor defined as ServiceProviderB and the hasSourceOfData defined as a union of the two classes representing sources that either are not authorized or are not trusted or vouched by a trusted party. Accordingly, the elements that must be placed in the condition part of the derivation rule can be represented as in Program 5.7. Program 5.7 represents the actor for the action of PolicyLDataAcceptance as object variable Y, and its context as Z. In Program 5.7, it has been shown that Y refers to all objects instantiated from the class &kaos_exp_actor;ServiceProviderB. Furthermore, Z has been defined as either the set of objects instantiated from the complement of (using <r2ml:isNegated>) the class AuthorizedPartyByServiceProviderB or the complement

1http://ontology.ihmc.us/SemanticServices/S-F/Example/index.html
Program 5.5: The KAoS policy that restricts the behavior of a broker agent to communicate only to the service providers that support X509 credentials
Program 5.6: The R2ML ObjectDescriptionAtom excerpt of the KAoS Policy Program 5.5 of the class TrustedPartyWithDelegateRights. Please note that, in the original KAoS version, these concepts have been represented using the <owl:complementOf> construct (see Program 5.5). R2ML models <owl:complementOf> by using its <r2ml:isNegated> element. Also to model the <owl:unionOf> element of OWL, we have taken advantage of R2ML's <r2ml:qf.Disjunction> which provides us with our expected meanings (i.e. disjunction or union of two extensional sets).

Also, as it can be seen in the KAoS excerpt of Program 5.5, there is an existential quantifier placed on the range of the hasSourceOfData property using <owl:someValuesFrom>. In our transformed R2ML rule, variable Z which represents the range for this property only appears in the condition of the rule. As a result, the quantifier over this variable is logically equivalent to an existential quantifier\(^2\) which makes our transformation fully compliant with the KAoS excerpt of Program 5.5. The appropriate derivation rule in this case can be obtained by placing the excerpt of Program 5.7 in the conditions and the excerpt of Program 5.6 in the conclusion of the derivation rule of Program 5.4.

\(^2\)Discussions with Dr. Adrian Guirca
Program 5.7: The R2ML ClassificationAtom excerpt of the KAoS Policy Program 5.5
5.2 Rei Policies to R2ML Derivation Rules

In this section we describe how the policy rules from Rei can be converted to R2ML. As an example consider a policy in which we need to prevent Graduate Students from printing in the Computing Science (CS) department. Program 5.8 shows how this policy is expressed in Rei\(^3\).

As one can see in Program 5.8 and based on what we described in Section 4.2, a Rei policy consists of a policy element representing a policy set, a deontic rule identifying the to-be-enforced policy rule, and a set of constraints that are placed on the definition of the policy set, the policy rule, the contexts, and the to-be-performed actions. In the example of Program 5.8 the policy set has been named CSPolicy and the deontic rule has been named Proh_StudentCSPrinting which intends to prohibit the graduate students from using the CS-Department printers.

Again, following our mapping rules of Chapter 4, a Prohibition policy in Rei can be mapped to a Prohibition class in our R2ML Policy Model. Thus, Proh_StudentCSPrinting is represented as an object instantiated from our Prohibition class, with its elements specialized according to the definition of the Rei policy. Figure 4.39 shows that our derivation rule is made by extracting concepts from both the policy and the deontic elements of the Rei policy. Program 5.9 shows how the ObjectDescriptionAtom, that is placed in the conclusion of a derivation rule, is made out of the elements of our sample Rei policy. The hasAction element of the ObjectDescriptionAtom gets its value from the action of the deontic element. The actor, however, is connected to the action using a ReferencePropertyAtom which is placed in the condition part of the derivation rule. The constraints in Rei are represented as ReferencePropertyAtoms that go into the conditions part of the derivation rule.

Note should be given that, the context of a Rei policy is also a constraint in the form of triples (see Program 5.8). In our mappings, the constraint over the context is also placed in the condition part of the derivation rule as a ReferencePropertyAtom. Yet, to not lose this

\(^3\)Taken from the website for the Rei policy language: http://www.cs.umbc.edu/~lkagal1/rei/examples/univ/deptpolicy.owl
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:owl="&owl;"
    xmlns:policy="&policy;" xmlns:action="&action;"
    xmlns:constraint="&constraint;" xmlns:deontic="&deontic;"
    xmlns:entity="&entity;" xmlns:univ="&univ;"
    xmlns:inst="&inst;" xmlns:deptpolicy="&deptpolicy;">

    <entity:Variable rdf:ID="#var1" policy:desc="A variable"/>
    <entity:Variable rdf:ID="#var2" policy:desc="A variable"/>

    <constraint:SimpleConstraint rdf:ID="#IsMemberOfCS">
        <constraint:subject rdf:resource="#var1"/>
        <constraint:predicate rdf:resource="#univ;affiliation"/>
        <constraint:object rdf:resource="#inst;CSDept"/>
    </constraint:SimpleConstraint>

    <constraint:SimpleConstraint rdf:ID="#IsStudentCSPrinting">
        <constraint:subject rdf:resource="#var2"/>
        <constraint:predicate rdf:resource="#rdf;type"/>
        <constraint:object rdf:resource="#inst;StudentCSPrinting"/>
    </constraint:SimpleConstraint>

    <constraint:SimpleConstraint rdf:ID="#IsGraduateStudent">
        <constraint:subject rdf:resource="#var1"/>
        <constraint:predicate rdf:resource="#rdf;type"/>
        <constraint:object rdf:resource="#univ;Graduate"/>
    </constraint:SimpleConstraint>

    <constraint:And rdf:ID="#GradStudentConstraint">
        <constraint:first rdf:resource="#IsGraduateStudent"/>
        <constraint:second rdf:resource="#IsStudentCSPrinting"/>
    </constraint:And>

    <deontic:Prohibition rdf:ID="#Proh_StudentCSPrinting">
        <deontic:actor rdf:resource="#var1"/>
        <deontic:action rdf:resource="#var2"/>
        <deontic:constraint rdf:resource="#GradStudentConstraint"/>
    </deontic:Prohibition>

    <policy:Policy rdf:ID="#CSPolicy">
        <policy:actor rdf:resource="#var1"/>
        <policy:context rdf:resource="#IsMemberOfCS"/>
        <policy:grants rdf:resource="#Proh_StudentCSPrinting"/>
    </policy:Policy>
</rdf:RDF>

Program 5.8: The Rei policy that prevents Graduate Students to print in the CS Department
Program 5.9: The R2ML ObjectDescriptionAtom for the conclusion part of our derivation rules

concept among other constraints of our R2ML rule, the value for hasContext in our model is replaced with a variable referring to this constraint. It helps to identify the proper constraint which is related to the context, as the variable in the subject of the ReferencePropertyAtom should be identical to the variable that hasContext of the ObjectDescriptionAtom refers to. Program 5.10 shows the excerpt of the R2ML code that goes to the conditions part of the derivation rule. To make a complete derivation rule, we can simply place Program 5.10 in the conditions part and Program 5.9 in the conclusion part of our derivation rule code, i.e. Program 5.4.

5.3 R2ML Derivation Rules to Rei Policies

So far we have shown the possibility of converting the policy rules from both KAoS and Rei to R2ML. The next step would be to show the possibility of converting the R2ML files back to Rei or KAoS. This in general leads us to our ultimate goal of having a general interchange framework to exchange policies between different business enterprises and business partners. In this section we chose the second example of Section 5.1 and show how a KAoS-driven
Program 5.10: The R2ML ReferencePropertyAtoms for the condition part of our derivation rules
CHAPTER 5. EXAMPLES OF TRANSFORMING BETWEEN POLICY RULES

R2ML rule can be converted to its Rei equivalent.

Programs 5.6 and 5.7 show the derived R2ML elements from the KAoS Program 5.5. As we mentioned earlier, the syntax of Rei is closer to what we have chosen for our R2ML representation of the transformations. Program 5.6 shows a Prohibition rule in R2ML as an extension of our R2ML Policy Model which is accordingly mapped to a Rei Prohibition. Through the mapping process, the Prohibition element of the R2ML rule is mapped to Rei's <policy:Policy> and <deontic:Prohibition> elements. Programs 5.11 and 5.12 illustrate the outcome of applying the transformations to the R2ML Programs 5.6 and 5.7. Finally, Program 5.13 clearly shows the mappings that happen between the similar elements of R2ML and Rei. The program shows that an ObjectClassificationAtom in R2ML is equivalent to defining an <entity:Variable> as well as assigning this variable to a type through a <constraint:SimpleConstraint> in Rei. As we mentioned in Chapter 4, in R2ML, the conclusion part of a derivation rule is a conjunction of atoms; but in Rei the conjunctive operator should be placed over every two constraints that are supposed to be conjuncted. So, <constraint:And> should be added as a connector while converting the R2ML expressions to the Rei equivalents.

A quantifier free disjunction element (i.e. qf.Disjunction) exists in R2ML to represent the disjunction or union of two different classes over a variable. This can simply be modeled as a <constraint:Or> in Rei. Also Program 5.13 illustrates that an ObjectDescriptionAtom in R2ML is divided into pieces while converting it to Rei, in order to make the appropriate matchings for the <deontic:Prohibition> and <policy:Policy> elements. Elements such as hasPriority and context should be converted to compatible elements in Rei's <policy:Policy>, while the R2ML objects for the action and the actor go into the <deontic:Prohibition> element. A close look at the programs shows how these pieces of R2ML and Rei, placed beside each other, are semantically close and carry the same intended meanings. This proves that our conversions do not demonstrate only the syntactical matching between the elements but it is actually the semantics that is carried through mapping the logics, identical concepts, and appropriate terms and atoms.
CHAPTER 5. EXAMPLES OF TRANSFORMING BETWEEN POLICY RULES 130

<rdf:RDF>

<entity:Variable rdf:ID="X"/>
<entity:Variable rdf:ID="Y"/>
<entity:Variable rdf:ID="Z"/>

<constraint:SimpleConstraint rdf:ID="IsServiceProviderB">
    <constraint:subject rdf:resource="#Y"/>
    <constraint:predicate rdf:resource="&rdf;type"/>
    <constraint:object rdf:resource="&kaos_exp_actor;ServiceProviderB"/>
</constraint:SimpleConstraint>

<constraint:SimpleConstraint rdf:ID="IsAcceptData">
    <constraint:subject rdf:resource="#X"/>
    <constraint:predicate rdf:resource="&rdf;type"/>
    <constraint:object rdf:resource="&kaos_exp_action;AcceptData"/>
</constraint:SimpleConstraint>

<constraint:SimpleConstraint rdf:ID="IsAuthorizedPartyByServiceProviderB">
    <constraint:subject rdf:resource="#Z"/>
    <constraint:predicate rdf:resource="&rdf;type"/>
    <constraint:object rdf:resource="&kaos_exp_actor;AuthorizedPartyByServiceProviderB"/>
</constraint:SimpleConstraint>

<constraint:SimpleConstraint rdf:ID="IsTrustedPartyWithDelegateRights">
    <constraint:subject rdf:resource="#Z"/>
    <constraint:predicate rdf:resource="&rdf;type"/>
    <constraint:object rdf:resource="&kaos_exp_action;TrustedPartyWithDelegateRights"/>
</constraint:SimpleConstraint>

<constraint:Or rdf:id="IsAuthorizedOrTrusted">
    <constraint:first rdf:resource="#IsAuthorizedPartyByServiceProviderB"/>
    <constraint:second rdf:resource="#IsTrustedPartyWithDelegateRights"/>
</constraint:Or>

</rdf:RDF>

Program 5.11: The Rei equivalent of Program 5.5 through R2ML - Part 1
<constraint:And rdf:id="ConjunctionCombination1">
  <constraint:first rdf:resource="#IsServiceProviderB"/>
  <constraint:second rdf:resource="#IsAcceptData"/>
</constraint:And>

<constraint:And rdf:id="ConjunctionCombination2">
  <constraint:first rdf:resource="#ConjunctionCombination1"/>
  <constraint:second rdf:resource="#IsAuthorizedOrTrusted"/>
</constraint:And>

<constraint:SimpleConstraint rdf:ID="SourceOfData">
  <constraint:subject rdf:resource="#X"/>
  <constraint:predicate rdf:resource="&kaos_exp_action;hasSourceOfData"/>
  <constraint:object rdf:resource="#Z"/>
</constraint:SimpleConstraint>

<deontic:Prohibition rdf:ID="Policy1_DataAcceptance">
  <deontic:actor rdf:resource="#Y"/>
  <deontic:action rdf:resource="#X"/>
  <deontic:constraint rdf:resource="#ConjunctionCombination2"/>
</deontic:Prohibition>

<policy:Policy rdf:ID="X509CommunicationPolicy">
  <policy:actor rdf:resource="#Y"/>
  <policy:context rdf:resource="#SourceOfData"/>
  <policy:grants rdf:resource="Policy1_DataAcceptance"/>
</policy:Policy>

</rdf:RDF>

Program 5.12: The Rei equivalent of Program 5.5 through R2ML - Part 2
CHAPTER 5. EXAMPLES OF TRANSFORMING BETWEEN POLICY RULES

Program 5.13: The Rei equivalent of Program 5.5 through R2ML - Concept Matching
5.4 R2ML Derivation Rules to KAoS Policies

For the last section in this chapter, we discuss how a Rei-derived R2ML policy rule can be mapped to KAoS. In section 5.2 we showed how a Rei rule, putting constraints over the grad students who can not access the printer, can be converted to R2ML. Here we discuss the possibility of converting this R2ML file to KAoS. As we mentioned several times in the previous sections, KAoS defines its constraints over different actions, actors, and contexts by using OWL restrictions which are placed over the properties of each class. Here we need to take the exact same approach to transform the atoms of R2ML to KAoS.

Let us start with looking at Program 5.14. On the right side we have the combined R2ML excerpt from Programs 5.9 and 5.10. We mentioned in Chapter 4 that to make a KAoS class (i.e. OWL class) out of R2ML atoms we have to determine the ReferencePropertyAtoms with identical subjects. The predicate for each ReferencePropertyAtom is equivalent to the property of the subject that is limited, and the object is equivalent to the values this property is restricted to. Also each ObjectClassificationAtom in R2ML represents a class, thus open occurrence of an ObjectClassificationAtom in the R2ML code we inherit a class from the class that the ObjectClassificationAtom introduces and then specialize it by evaluating the appropriate restrictions.

Program 5.14 shows that there are two classes that should be created while making the target KAoS class, the first one represents the extensional set for the class &uni;Graduate and the second one represents &inst;StudentCSPrinting. According to our definitions in the previous paragraph, we should have two classes that also represent these extensional objects. The GraduateActor class is inherited from the class &uni;Graduate, but we are not done with the definition of the GraduateActors class. To find the appropriate restrictions that limit the behavior of the objects extended from this class we look for all the ReferencePropertyAtoms that in their subjects have a name similar to the one introduced by the ObjectClassificationAtom. For example in Program 5.14, var1 as the object variable instantiated from the class &uni;Graduate is also the subject for the second ReferencePropertyAtom in the same code. We take the property (defined as the predicate in the ReferencePropertyAtom) and restrict it to the values defined by its object. If the object for the ReferencePropertyAtom is a variable, in our KAoS definition we replace this instance
with the class that it has been instantiated from, otherwise the object itself will be placed.

Having another look at Program 5.14, one can see that for the first KAoS class that we generated, i.e. GraduateActor, the restriction is placed over an OWL instance because the object of the ReferencePropertyAtom is &inst;CSDept which is an instance. However for the second class, using the same mapping conventions as above, the deontic:actor property is restricted to the class GraduateActors because in the corresponding R2ML ReferencePropertyAtom the value for the object is the variable var2.

As we mentioned earlier, the Prohibition class from our R2ML Policy Model is mapped to NegAuthorizationPolicy with its <policy:control> element pointing to the action that the hasAction element in our R2ML model refers to. However, as in our model var2 is the value for this property, initially instantiated from the class &inst;StudentCSPrinting and specialized as Proh.StudentCSPrintingAction in our transformation, the equivalent value for var2 is an object that is instantiated from this specialized class. Program 5.14 shows that Proh.StudentCSPrintingActionObj is the object instantiated from this class and used as the to-be-controlled action in place of var2 in our R2ML model. As a result, the excerpt of code in the right side of Program 5.14 is the KAoS equivalent for the Rei Program 5.8.

5.5 Summary

In this chapter we practically showed how the concepts from one policy language can be converted to the concepts of other policy languages. Our transformations have been implemented using W3C's XSLT [107] and can be found in [117].

In Section 5.1 we exemplified two KAoS policy languages and showed how they can be transformed to R2ML. In Section 5.2 the same approach was taken to show the possibility of converting the policy rules from Rei to R2ML. In both cases we tried to show that the chosen R2ML elements for the purpose of transformations have been considered with a lot of care in order to make the target policy rule as close as possible to the source policy rule. In Section 5.3 we chose one of the two examples of the first section (the more challenging one), and showed how this policy can be converted to Rei. In Section 5.4, the same approach was
Program 5.14: The RML equivalent of Program 5.8 through R2ML - Part 2
applied to the example of Section 5.2 and the example of this section was converted to KAoS.

We have validated our transformation results by using OWL, XML Schema, and RDF validators. It shows that our transformations are syntactically sound, but we still need to conduct more research on how efficient the transformations are when we use them in real practical domains, how effective the converted policies behave in protecting resources, and how information loss affects the desired results. In the next chapter we slightly argue the concepts that have difficulties to be covered while applying the transformations. Furthermore, we argue how these transformations can be converted to other rule languages and be used in other domains.
Chapter 6

Analysis of the Policy Interchange Framework

To this point we have clarified the main intentions for mapping between different policy languages, we have introduced two policy languages, namely Rei and KAoS, and we have investigated their internal constructs. We have also compared different elements of these policy languages and have shown that they more or less cover the same concepts. Exploiting the similarities between the concepts of KAoS and Rei, we showed how R2ML as an intermediary policy language with its abstract syntax can be employed to play as a bridge between the concepts of KAoS and Rei. We also showed some concrete examples of exchanging the policies between KAoS and Rei by transforming them from one policy language to R2ML and then from R2ML to the other policy language.

In this chapter we analyze the transformations that we have explained so far, we show how these transformations can be used together with other transformations that have been already developed for other rule languages based on derivation rules, and we discuss the possibility of using them to convert our policy-driven R2ML rules to the rule languages that are supported by those policy languages (cf. Section 6.1).

Moreover, in Section 6.2, we analyze some of the differences between KAoS and Rei and explain how these differences affect the transformations. We highlight that R2ML can cover (or can be extended to cover) all the concepts of different policy languages, but it does not
guarantee that the concepts are then transformable to other policy languages.

Section 6.3, as the last section in this chapter, describes the modifications that we have found necessary to extend R2ML. In this chapter, we also describe the possibility of modeling policy rules using other constructs of R2ML such as Integrity Rules.

6.1 Compatibility of our Transformations with other R2ML Transformations

As we already stated, R2ML is supposed to be an intermediary language, to/from which other policy languages can be transformed. To achieve this goal several transformations have been defined and developed between R2ML and other policy languages by mapping the rules from these languages to one of four main types of rules that R2ML supports, i.e. derivation rules, integrity rules, production rules, and reaction rules (cf. Section 3.2.7).

Table 6.1 shows the current transformations that have been developed between R2ML and other rule languages as well as their corresponding R2ML rules which have been used. As our transformations to R2ML are converted to its subset of derivation rules, it would be a good practice to see how efficient and meaningful these rules can be transformed to other rule languages that are derivable from R2ML derivation rules. At the time of writing this dissertation, RuleML rules, Jess rules, Jena rules, and F-Logic rules can be obtained from R2ML derivation rules. In this section we briefly review the conversions above and show how efficient the transformations are. These transformations are available at [86].

<table>
<thead>
<tr>
<th>R2ML</th>
<th>RuleML</th>
<th>Jess</th>
<th>F-Logic &amp; F-Logic XML</th>
<th>JBoss</th>
<th>Jena</th>
<th>SWRL</th>
<th>OCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derivation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transformation Language</td>
<td>XSLT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Transformations between R2ML and other rule languages
6.1.1 R2ML to RuleML

We mentioned in Section 2.5.1 that RuleML is most likely the closest attempt to R2ML. Transformations from R2ML to RuleML and back has already been developed by our collaborators at Technical University of Brandenburg at Cottbus. Also online transformers for RuleML to R2ML\(^1\) and R2ML to RuleML\(^2\) have been already developed. To examine the possibility of using these transformations we ran them over our R2ML excerpts. Programs 6.1 and 6.2 show the RuleML model obtained from our 5.5 and 5.8 respectively through a conversion from Programs 5.9 and 5.10.

The mappings provided in the left column of Programs 6.1 and 6.2 clearly show that the generated RuleML excerpts represent promising equivalents of the KAoS and Rei policies that have been provided in the right columns of the two tables. Please note that the RuleML codes have not been generated from the KAoS policy or the Rei policy, but they have been generated from R2ML. The obtained results can be used with any other reasoner or program that can work with RuleML rules. Considering an application that supports the definition of constraining rules in RuleML, its developer can easily integrate any policy from KAoS or Rei into their applications by applying our transformations to the original KAoS or Rei policies twice.

6.1.2 R2ML to Jena

JenaRules package includes a general purpose rule-based reasoner which is used to implement both the RDFS and OWL reasoners but is also available for general use [18, 89]. This reasoner supports rule-based inference over RDF graphs and provides forward chaining, backward chaining and a hybrid execution model. A rule set is simply a list of rules. The hybrid execution model allows forward chaining rules to invoke backward chaining rules. The forward rules work incrementally, including incrementally asserting or removing backward rules in response to the data changes. The hybrid execution makes available both forward chaining rules and backward chaining rules in the same session.

\(^1\)http://oxygen.informatik.tu-cottbus.de/translator/RuleMLtoR2ML/

\(^2\)http://oxygen.informatik.tu-cottbus.de/translator/RuleML
Program 6.1: The RuleML equivalent of Program 5.5 through R2ML and the one-to-one mappings
Program 6.2: The RuleML equivalent of Program 5.8 through R2ML and the one-to-one mappings
There have been transformations defined from R2ML to Jena \(^3\) that support converting R2ML derivation rules to Jena rules. Again we conducted an experiment over our R2ML derivation rules derived from Rei policy of Program 5.8 by using the R2ML to Jena transformer. Program 6.3 shows the details of mapping the elements from this Rei policy rule to Jena. Having a close look to the Jena rule at the left side of Program 6.3 shows that this rule generates a Proh_StudentCSPrinting object with an action defined as ?var2 and a context defined as ?X. It also shows how the variables are defined and the appropriate values for them are allocated.

Jena uses \texttt{rdf:type} to refer to the class that a variable or an object belongs to. Similarly, Rei uses the same approach to define the type of an object or a variable. It is interesting to note that, the derived Jena rule of Program 6.3, although generated from our R2ML rule, is it really close to how Rei defines the instantiations of variables from classes, which in turn, makes the whole structure of the two rules in Jena and Rei really close to each other.

However, trying to use the KAoS policy example with Jena was not as straightforward. We executed the Jena transformation over the R2ML policy from the KAoS Program 5.5, which resulted in the following error message by the transformation engine.

\texttt{r2ml:qf.Disjunction is not supported by Jena Rules!}

This clearly shows the problems that we have mentioned several times throughout this dissertation. The feasibility of the transformations is in a direct relationship with the concepts that each language supports. The transformations are applicable only to the concepts that are shared between the languages involved in the process of transformation. If two languages don’t share common critical concepts, the obtained results would be erroneous and the level of accuracy would decrease. We mentioned in Chapter 5 that \texttt{qf.Disjunction} was employed in our R2ML model to cover the \texttt{union0f} concept in OWL and KAoS. Apparently the transformation of \texttt{r2ml:qf.Disjunction} is not supported by the developed transformer from R2ML to Jena.

\(^3\)http://oxygen.informatik.tu-cottbus.de/translator/Jena2/
CHAPTER 6. ANALYSIS OF THE POLICY INTERCHANGE FRAMEWORK

Program 6.3: The Jena rule equivalent of Program 5.8 through R2ML and its one-to-one mappings
6.1.3 R2ML to F-Logic

For this section we work on a different example from the ones above. The example has been taken from [68] in which a complete description of the problem can be found. The example considers a buyer and a seller, with two different policy languages, who need to negotiate their policies and preferences. In this scenario, the buyer wants to buy a device from an online web shop (see Figure 6.1).

![Diagram](image)

**Figure 6.1: An eCommerce scenario for exchanging policies**

The buyer employs a software called *BuyerAgent* that functions as the automated negotiating agent, while the web shop server employs another software called *SellerAgent* as the automated negotiating agent. The policies for the buyer and the seller describe who they trust, how, and what purposes for. The negotiation is based on these policies specified as rules and credentials for SellerAgent and BuyerAgent. The policies can be interchanged (i.e., negotiated and disclosed) in this process to establish the trust between the involved parties. Assuming that BuyerAgent and SellerAgent use different languages for defining and evaluating rules (policies), e.g. KAoS by the BuyerAgent and F-Logic-based PeerTrust rules by the SellerAgent, R2ML needs to be employed for enabling the rules to be transformed between the two software systems and to facilitate the process of negotiation.
Referring to Figure 6.1, when the buyer wants to buy a product and clicks on the “buy it” button at the online web shop (step 1), its BuyerAgent takes over and sends a request to the online web shop (step 2). SellerAgent receives the request, retrieves its policies stored in its rule repository (step 3 and 4) and sends parts of its policy as sets of rules back to the BuyerAgent (step 5). The policy states: “If a buyer provides the delivery information (address, postal code, city, and country) then he must provide his credit card as well”. SellerAgent stores its PeerTrust policies in terms of F-logic rules in its rule repository. The rule for the policy above has been shown in Program 6.4.

```plaintext
provideCreditCardActionObj:Obligation[
  hasPriority->10;
  hasSiteOfEnforcement->targetSite:SiteOfEnforcement;
  hasObligationConstraint->finishBeforeDeliveryInfoEnd:FinishBeforeRefActionEnd;
  performedBy->?varBuyer; hasObjectContext->?varCC]
<-?
?varBuyer:Buyer AND ?varCC:CreditCard AND
?varDeliveryInfo:DeliveryInformation AND
deliveryReqActionObj:DeliveryReqAction AND
deliveryReqActionObj:DeliveryReqAction
  [performedBy->?varBuyer] AND
deliveryReqActionObj:DeliveryReqAction
```

Program 6.4: The F-Logic rule for the Seller-Buyer scenario

Establishing an agreement between BuyerAgent and SellerAgent about the language in which they share their rules is the ideal case which seems to be impossible due to the variety of resources, broker agents, and Web rule languages that they may use. An easier way is to send the rules (policies) in the language in which they have been originally defined, i.e., F-Logic for the SellerAgent in our scenario. In case there is no understanding about F-Logic by the BuyerAgent or no knowledge about KAoS by the service provider, the communication would fail. So, either the BuyerAgent or the SellerAgent must be able to exchange the policy to an equivalent KAoS policy (rule). By using R2ML as the interchange language, SellerAgent does not send the policy in F-Logic format, but it translates it to the R2ML interchange format and sends it out (step 5. in Figure 6.1). BuyerAgent retrieves and translates the R2ML rule to the format it uses, i.e., KAoS (step 6. in Figure 6.1). The KAoS policy result of this transformation can be similar to the policy defined in Program 6.5.
Program 6.5: The KAoS rule for the Seller-Buyer scenario
After retrieving this policy, BuyerAgent and SellerAgent interchange some more messages to identify the level of trust in each other. Once the desired level of trust is established and BuyerAgent sends its delivery information, it also sends credit card information, again in the format of R2ML (step 7. in Figure 6.1). SellerAgent retrieves the information and translates it from the R2ML format into the F-logic format (step 8. in Figure 6.1). Thereafter, SellerAgent sends a message to BuyerAgent confirming the completion of the purchase (step 9. in Figure 6.1).

Choosing the appropriate constructs from R2ML and applying them to the transformations from both sides, we get the target R2ML excerpt that can play as a bridge between the two policy languages. Programs 6.6 and 6.7 show some parts of these transformations between R2ML, F-Logic, and KAoS respectively. As a result, the transformations prove the possibility of exchanging rules from KAoS to F-Logic.

6.2 Information Loss during Policy Transformations

In Section 6.1.2 we discussed that for the KAoS policy rule of Program 5.5, it was impossible to provide meaningful transformations to Jena, mainly because Jena could not support \texttt{r2ml:qf.Disjunction} in its policy rules. It happens in most transformations between different policy languages due to the differences in their underlying concepts. Rei and KAoS, as two different languages, have also their own constructs and elements to describe the policies and the concepts. It is not reasonable to consider that every single concept from KAoS can be mapped to Rei or vice versa.

In Section 4.2.7 we showed the metamodels for the \textit{Speech Acts} in Rei. Speech Acts have been introduced in Rei without any equivalent concept available for them in KAoS. Also in Section 4.2.8, we defined the metamodels for some of the concepts that have been defined in KAoS without any similar concepts available for them in Rei. We claimed earlier that R2ML has the suitable constructs to support the concepts of these policy languages. This is mainly because R2ML has been designed in a much more abstract level than Rei or KAoS. Looking back at Figure 3.5 that shows different levels in which rules can be defined, we can classify Rei or KAoS as platform specific languages while R2ML is in the cutting edge of platform independent and computation independent models. Thus R2ML rules support
Program 6.6: The one-to-one mappings between F-Logic and R2ML for the Seller-Buyer scenario
Program 6.7: The one-to-one mapping between KAoS and R2ML for the Seller-Buyer scenario
more broad meanings than KAoS or Rei. Furthermore, R2ML has considered properties for computational logic and description logic which again enables it to share the concepts between these two domains.

In our model, regardless of the policy language that is being used, the concepts can be mapped to R2ML. For example Speech Acts from Rei or the concepts for Team and Domain from KAoS can be covered by extending our R2ML Policy model of Figure 4.32, yet it should be investigated how these concepts are effective and useful when they are shared between different policy languages. Extending our R2ML Policy Language by the concepts from different policy languages under study, does not make the whole model more generic, but it reduces the expressivity when coping with the abstract notation of different policy languages. Extensions to a policy model should be done carefully and after deeply reviewing the pros and cons of the newly added concepts. One can see in our definition of the R2ML Policy Model that we have not extended the model to cover the concepts that Rei introduces for Modality, Rule Precedence, Behavior, and etc. It is mainly because we are not convinced about the necessity of representing those concepts in defining policies. Although these concepts are definitely helpful in regulating the behavior of the reasoning engine, they are not the key concepts to define the policies and a policy rule can survive without supporting these elements in its definition. This is what the transformations to R2ML prove.

To be able to check the feasibility of the transformations in protecting the resources, a deeper analysis should be conducted. The most promising approach would be to deploy the policies on different resources and broker agents and get them to communicate. The level of jeopardizing the resources and the to-be-protected contents should be carefully examined. Policies are the critical rules in a business system which regulate the behavior of the system. Any flaw or miss-interpretation of the policies may result in the noncompensatable loss of data and information. This further necessitates a detailed examination of the policies while working on the level of policy exchange. The practical analysis of the transformed policies is part of the upcoming research project that will be conducted in the Laboratory for Ontological Research (LORe) at Simon Fraser University.

4http://lore.iat.sfu.ca/
6.3 Rules and Extensions for R2ML to be used as a Policy Rule Language

This dissertation demonstrated the possibility to exchange the policy languages by 1) high level meta-model representation of their concepts, 2) capturing the semantics of the concepts, 3) identifying the similarities and dissimilarities of the modeled concepts, and 4) moving to the more concrete syntax of the languages and applying the transformations. All the above steps are compliant with the required steps in software design using Model Driven Engineering techniques (MDE) [54]. Using MDE techniques to provide the transformations significantly assists with having valid models for the source and the target languages. Furthermore, while using MDE, we are able to detect the inconsistent or missing constructs of different languages.

During working with policies we discovered some points of improvement for R2ML which can add to its efficiency. First of all, we found it is important for rules to have a value as an indicator for their priority. Having a set of rules, as in R2ML derivation or integrity rule sets, we need to provide an indicator of how one rule in this series can be chosen over the other ones. It especially helps with cases where more than one rule might be applicable to a certain situation. Although this issue matters mainly at the level of rule enforcement, where the enforcement engine needs to select among a set of applicable rules, and knowing that the current versions of R2ML are not supposed to be used at any enforcement level, dealing with rule exchange for cases where the source or the target rules need to have priorities makes it necessary to have priority indicators for the rules.

Another point that seems to be open to extend in R2ML, is the possibility to use quantifiers to express cardinality. While working with KAoS, we realized that this language supports min_cardinality and max_cardinality. However, these quantified formulas are not possible to be used in the condition of a derivation rule set as the elements of the condition part are considered quantifier free and should be universally quantified. An extension to R2ML such that cardinalities can be supported in the condition of the derivation rules is also a possible extension to the new versions of R2ML.

Further to this, there have been long discussions on whether derivation rules or integrity
rules should be used to model the policies. Looking back at the definitions of integrity and derivation rules in Section 3.2.7, we see that integrity rules define that something must necessarily hold, or it should hold, while derivation rules carry a derivative meaning representing a set of new conclusions based on the presented facts. Both of these types of rules can be used to define the policies depending on the type and the purpose of the policy (i.e. authorization, authentication, Quality of Service, etc.).

A policy similar to the one in Section 6.1.3 can be described with integrity rules more meaningfully as compared to its modeling through derivation rules. Program 6.8 shows how Program 6.4 can be modeled using R2ML integrity rules. The representation of this policy (or integrity rule) in natural language would be: "As per request of delivery, by an arbitrary buyer, containing delivery information, s/he MUST also provide her/his Credit Card information in the same request." One can see that the definition of the rule described in Section 6.1.3 is a lot more straightforward in Program 6.8, as compared to the R2ML derivation rules we obtained in Programs 6.6 and 6.7. Yet it should be noted that the obtained Program 6.8 is not easily convertible to KAoS or Rei. On the other hand, the authentication and authorization policies have a derivative meaning indicating that the permission or prohibition is concluded upon satisfaction of the credentials. Further research is needed in this area to achieve an automatic and promising selection between derivation and integrity rules when modeling policies.

6.4 Summary

In this chapter we analyzed some of the issues we faced during providing the transformations. In Section 6.1 we showed how R2ML as an interchange language can perfectly play as the intermediary language to transform the concepts from one language to another. We tried our R2ML models for KAoS and Rei with the transformations for RuleML, Jena, and F-Logic and showed that the obtained results were satisfactory.

In Section 6.2 some of the differences between KAoS and Rei were explained and the possibility to exchange our R2ML Policy Model to cover these concepts was argued. We mentioned that, although it is possible to extend R2ML such that all the concepts from all policy languages can be captured, this extension would not result in a comprehensive
CHAPTER 6. ANALYSIS OF THE POLICY INTERCHANGE FRAMEWORK

Program 6.8: The Integrity Rule representation for Program 6.4

<table>
<thead>
<tr>
<th>r2ml:DeonticIntegrityRule</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2ml:constraint</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>r2ml:UniversallyQuantifiedFormula</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>r2ml:ReferencePropertyAtom</td>
</tr>
</tbody>
</table>

Program 6.8: The Integrity Rule representation for Program 6.4
language but it adds to the complexity of the transformations. We do not consider the generality of our policy language as its capabilities to capture everything from everywhere, but we consider it general if it can be a flawless bridge between the key and critical concepts of different policy languages.

Finally, in Section 6.3 we discussed how R2ML should be extended to cover more critical concepts in policy languages, and also we had a brief comparison of integrity and derivation rules to examine whether one of these rules can be used in place of the other one, and how. A detailed investigation of these possible concepts is under review by our R2ML research team.
Chapter 7

Conclusions

Our project declared the possibility of providing transformations between different policy languages. By adopting policy languages for the purpose of transformation, we showed that R2ML is not just a language to support abstract domain independent rule languages such as F-Logic or RuleML, but domain specific rule languages (including policy languages) can also be transformed by making use of R2ML. Furthermore we showed that existing transformations from R2ML to other languages can be executed over our obtained R2ML results which then enable us to evaluate the policies through different policy engines. In this chapter we discuss the contributions of this research work to the community of Semantic Web and Rule Languages and we elaborate on future directions of the research.

7.1 Contributions

This project should be considered as a collaborative work among the research groups from the Laboratory for Ontological Research (LORe) from Simon Fraser University, Technical University of Brandenburg at Cottbus, Germany, and University of Belgrade at Belgrade. Throughout the project, the research at the LORe laboratory of Simon Fraser University was supported by Canada’s NSERC-funded LORNET Research Network, while the research of Brandenburg University of Technology at Cottbus was supported by the EU IST-funded REWERSE Network of Excellence.

Our attempts to provide transformations between different policy languages showed the
feasibility of policy exchange between different service providers and broker agents by employing Model Driven Engineering (MDE) approaches. This led us to provide meaningful transformations between different policy languages to be used as the initial step towards a general modeling language for policies and to integrate this policy modeling approach with other features that MDE offers, including rapid prototyping of software products. Policy RuleML [82] is a similar attempt in the area of policy transformation. It aims at making RuleML a semantic interoperation vehicle for heterogeneous policies and protocols on the Web. However, to the best of our knowledge there is no proposed work done by this group and thus our attempt in exchanging policy rules between different entities is the first practical attempt in this area.

R2ML can be considered as the most general existing rule markup language. Our attempts in making it compatible with rule languages of lower abstraction level would add to its generality and comprehensiveness. Our approach provided mappings between R2ML and, KAoS and Rei as the two most known context-based policy languages. Although KAoS and Rei are both taken from academia and there is not very much of industrial support for these policy languages, they represent the migration to the new generation of policy languages in which better care is given to the actions that are executed on the resources rather than just to the users of those actions (as it is the case with the current role-based policy languages). Additionally, KAoS and Rei have more conceptual similarities than a role-based and a context-based policy language can possibly have.

For our transformations, we provided practical implementations showing the possibility of converting between these policy languages. The XSLT transformations are proofs for the feasibility of our mappings between the concepts of these policy languages. The obtained results have been syntactically validated and proved correct. Nevertheless, a better evaluation should be conducted to investigate the pros and the cons of transforming between policy languages when they are applied to the real world scenarios. These evaluations are subject to the future work in the project.

We also exported the outcomes of our mappings to the other transformers available from R2ML to other rule languages, e.g. Jena, RuleML, and F-Logic, and showed how these conversions can happen from KAoS or Rei to any of the languages above through using R2ML.
It proves the possibility of using the outcomes of our mappings together with the existing transformations for the purpose of inter business rule exchange as one of the targets followed by RIF.

Our approach enabled us to fully investigate the structure of the existing policy languages, mainly KAoS and Rei, and compare their similarities and differences. It not only clarified how these languages can be mapped to each other through using R2ML, but also provided us with a deep understanding of how different elements of these languages work together, and what makes policy languages appropriate to be used in different concepts. The key concepts were studied in detail in order to obtain results that are both logically valid and computationally sound.

The study of these languages also provided us with the possibility to recognize the concepts that R2ML requires in order to support the transformations between low level domain specific policy languages. It resulted in identifying and introducing elements that can further add to the generality of R2ML and make it an appropriate language to be adopted world wide for the purpose of inter enterprise communication.

Having all the discussions above, we can bullet point the contributions of this research work to the area of Semantic Web, Web Engineering, Model Driven Engineering and Rule Interchange as:

- A detailed review and analysis of the research areas relevant to the objectives of our research, including but not limited to ontology design and definition, Semantic Web and Model Driven Engineering, distributed trust and policy management, and rule languages and their purposes on the Web.

- Reviewing the existing policy languages for distributed systems and networks, investigating their conceptual intentions and analyzing the existing software, engines, and tools that can work with these policy languages.

- Providing the meta-model definition of KAoS and Rei as the two most known context-based policy languages, as opposed to the traditional role-based policy languages that work at the level of the actors of a system. The meta-models follow the recent UML 2.0 notation specification.
• Comparative analysis of the two policy languages, i.e. KAoS and Rei, through using their meta-models.

• Adopting QVT’s graphical notation to define the transformations between different elements of the policy languages and R2ML.

• Developing transformations between the concepts of KAoS and Rei using XSLT.

• Adopting transformations from R2ML to other rule languages including Jena, RuleML, and F-Logic and applying them to our R2ML definitions for KAoS and Rei to be used in real rule exchange application domains.

• Recommendations on how to further extend R2ML to support languages with lower levels of abstraction, i.e. Business Process Languages, Policy Languages, XSB-Prolog rules, etc.

7.2 Future Work

As we mentioned in the previous section, the detailed study of different policy languages gave us a broad understanding of the requirements for a policy language to be adopted by the community of system developers. On the other hand, model driven and software engineering approaches widely promote the use of models to define the workflow of a system, regulate its behavior, and analyze its efficiency at the design level rather than at the implementation level.

Additionally, the nonexistence of a policy modeling language to be integrated with other modeling languages for software product lines obviously is being felt by the community of software engineering and model driven engineering. An appropriate modeling language, besides having a set of graphical notations which visually represent its constituents, requires an underlying language that can model its concepts clearly and unambiguously. Policy languages as an extension of rule languages can be neatly modeled with more abstract rule languages.

R2ML, as the most general rule markup language to the date, can be played as the
appropriate language to be used for the purpose of defining a general policy modeling language. Our long term mission would be to migrate towards having a *General Policy Modeling Language (GPML)* that can support the definition of policies through using models and advanced model driven concepts. At the moment, *Strelka* [98], a UML-based visual rule modeling tool, has been developed that supports defining R2ML through using URML [101]. Our goal will be to extend both the definition of R2ML and the graphical notation that Strelka supports so that we can obtain a modeling language for policies.

Summing up, we can define the future goals followed in this project as follows:

- Improvements and extensions to the R2ML in order to make it compatible with domain specific languages, and in particular policy languages:
  - Deciding and justifying the distinctions between the policy rules that should be defined as integrity rules and those that have to be defined using derivation rules.
  - Supporting some level of uncertainty for R2ML atoms to cope with the situations where it is not possible to collect all the credentials from the counterpart entity.
  - Supporting priorities over the rules of a rule set, adding attributes to the RuleSet element that allow further reasoning over the rules of a set.

- Extending Strelka and the current graphical notations for URML to support the key concepts of policy languages in order to move towards a General Policy Modeling Language.

- Extending R2ML to support the concepts for policy elements. This extension will be considered as the underlying language that can entail the policy concepts which are expressed graphically through using Strelka or the underdeveloped R2ML modeling tools.

- Transformations from the GPML to the most known policy languages by the community of policy, including but not limited to XACML, KAoS, and Rei.

- Providing reverse engineering facilities, i.e. tools to convert any of the abovementioned policy languages to GPML and then to its graphical notation, representing the meta-model for the policy using the GPML's graphical notation.
Bibliography


BIBLIOGRAPHY


