Design and Implementation of a Distributed Rule-Based Query System Supporting Conference Organization Committees

by

Chaudhry Usman Ali

Bachelor of Computer Science, University of Punjab, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science, UNB

In the Graduate Academic Unit of your GAU

Supervisor(s): Dr Harold Boley
Dr Weichang Du
Examine Board: name1, degree, department/field, Chair
name2, degree, department
External Examiner: name, degree, department/field, institution

This thesis, dissertation or report is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK
Month, Year of submission to Graduate School
©Chaudhry Usman Ali, Year of graduation
Dedication (if any)

Start writing from here. This is optional.
Abstract

Conference organization involves a multitude of procedures consuming much time and effort of their organization committees (OCs). Conference organizations systems attempt to alleviate the burden of repetitive tasks through the (partial) automation of organizational processes. This thesis is focused on the design and implementation of automated query answering about a conference, retrieving and deriving OC-related information for use by (other) OC members, (candidate) PC members, (prospective) authors, as well as (potential) partners, sponsors, and participants. The Rule Responder framework is instantiated to a distributed rule-based system relieving OC members from answering such routine requests. Each team of co-chairs from the symposiums OC is supported by a Personal Agent (PA) that uses a local knowledge base containing co-chair facts and rules to answer queries for which the co-chairs are responsible. The Organizational Agents (OAs) act as a single point of entry to the managed sets of local PAs to which requests from the External Agent (EA) are disseminated. The user interacts with the system through an EA which can use an HTTP port to which post and get requests
can be sent via a Web form. The user requests are delegated to PAs by the organizational agent (OA) for answering the query. The responses are then forwarded to the EA for display. The designed query-answering architecture has been implemented and deployed in the SymposiumPlanner-2012 use case supporting the RuleML-2012 Symposium. General design principles and implementation techniques for future conference planners are distilled from the lessons learnt from this use case.
Acknowledgements (if any)

Start writing here. This is optional.
Table of Contents

Dedication ii
Abstract iii
Acknowledgments v
Table of Contents x
List of Tables x
List of Figures xi
Abbreviations xii

1 Introduction 1
   1.1 Overview ......................................................... 2
   1.2 Thesis Objective and Methodology ............................... 7
   1.3 Thesis Organization ............................................... 10

2 Background 11
   2.1 The Semantic Web ................................................. 11
# Rule Languages and Tools

## 3 Rule Languages and Tools

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Rules</td>
<td>17</td>
</tr>
<tr>
<td>3.2 XSLT</td>
<td>18</td>
</tr>
<tr>
<td>3.3 Rule Languages</td>
<td>19</td>
</tr>
<tr>
<td>3.3.1 RuleML</td>
<td>19</td>
</tr>
<tr>
<td>3.3.2 POSL</td>
<td>19</td>
</tr>
<tr>
<td>3.4 Semantic Web Rule Languages, Rule Engines and Tools</td>
<td>20</td>
</tr>
<tr>
<td>3.4.1 Prova</td>
<td>20</td>
</tr>
<tr>
<td>3.4.2 Reaction RuleML</td>
<td>32</td>
</tr>
<tr>
<td>3.4.3 OO jDREW</td>
<td>33</td>
</tr>
</tbody>
</table>

# Rule Responder

## 4 Rule Responder

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Approaches to Knowledge Representation</td>
<td>40</td>
</tr>
<tr>
<td>4.2 Distributed Architecture Approaches</td>
<td>40</td>
</tr>
</tbody>
</table>

vii
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1</td>
<td>Distributed System</td>
<td>41</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Hierarchical (Star Topology, Client-Server-Like Architecture)</td>
<td>41</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Networked (Full/Partial Mesh Topology, Peer-to-Peer-Like Architecture)</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>Rule Responder Architecture</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>Communication Middleware</td>
<td>45</td>
</tr>
</tbody>
</table>

5 SymposiumPlanner: Rule-based System 48

5.1 Interaction Between Organizational and Personal Agents ... 48
5.2 Task Responsibility Matrix ........................................... 50
5.3 Global Knowledge Base for Virtual Organization ............... 56

5.4 FOAF ........................................................................ 62
5.5 Locally Distributed Knowledge Bases for Personal Agents ... 65

5.5.1 Processing of Facts and Rules in a Personal Agent Profile 74
5.5.2 Illustrative Example .................................................. 77
5.6 Mule ESB ................................................................. 81

6 Case Study: SymposiumPlanner-2012 Query Based Conference Management System 90

6.1 Distributed Rule System vs. a Centralized Rule System .... 93
6.2 Organizational Agent ..................................................... 95
6.3 Responsibility Assignment Matrix .................................... 99
6.4 Personal Agents ......................................................... 100
6.5 Performatives ........................................... 108
6.6 Agent Communication Protocols ....................... 110
6.7 System Evaluation ...................................... 112

7 Conclusions ............................................. 118
7.1 Contributions .......................................... 118
7.2 Future Work ........................................... 119

Bibliography .................................................. 124

A System Resources ........................................ 125
A.1 User Client ............................................. 125
A.2 Prova Knowledge Bases ............................... 125
   A.2.1 Super-Organizational Agent (RuleML-2012 RuleML Structure) ......................... 125
   A.2.2 Sub-Organizational Agent (RuleML-2012) ........................................ 126
   A.2.3 Sub-Organizational Agent (RuleML Structure) ................................... 126
A.3 Responsibility Assignment Matrix .................... 126
   A.3.1 Sub-Organizational Agent Responsibility Assignment Matrix ......................... 126
A.4 Personal Agent Profiles ................................ 126
   A.4.1 General Chair ..................................... 126
   A.4.2 Publicity Chair .................................... 127
   A.4.3 Program Chair ..................................... 127
Glossary

List of Tables

6.1 Role Assignment Matrix ............................... 99

List of Figures

2.1 Semantic Web Architecture ............................ 12
3.1 Reaction Rule Syntax ................................. 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Reaction Rule Message</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>OO jDREW Example</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>Star Topology</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Mesh Topology</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Partial Mesh Topology</td>
<td>43</td>
</tr>
<tr>
<td>4.4</td>
<td>Rule Responder Architecture</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>Role Assignment Matrix</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Media Partner Query</td>
<td>78</td>
</tr>
<tr>
<td>5.3</td>
<td>Media Partner Query Response</td>
<td>79</td>
</tr>
<tr>
<td>5.4</td>
<td>Mule ESB</td>
<td>82</td>
</tr>
<tr>
<td>5.5</td>
<td>Mule Architecture</td>
<td>84</td>
</tr>
<tr>
<td>6.1</td>
<td>SymposiumPlanner 2012</td>
<td>94</td>
</tr>
<tr>
<td>6.2</td>
<td>PAs in Sub-OAs SymposiumPlanner</td>
<td>96</td>
</tr>
<tr>
<td>6.3</td>
<td>Publicity Chair Servlet</td>
<td>116</td>
</tr>
<tr>
<td>6.4</td>
<td>Servlet Response</td>
<td>117</td>
</tr>
</tbody>
</table>
List of Symbols, Nomenclature or Abbreviations

RuleML - RuleML Markup Language
OO RuleML - Object Oriented Rule Markup Language
FOAF - the Friend Of A Friend project
RDF - Resource Description Framework
RDFS - Resource Description Framework Schemas
POSL - Positional-Slotted Language
XSLT - The Extensible Stylesheet Language Transformations
XML - Extensible Markup Language
jDREW - java Deductive Reasoning Engine for the Web
OO jDREW - Object Oriented java Deductive Reasoning Engine for the Web
WWW - World Wide Web
URI - Universal Resource Identifier
HTTP - Hypertext Transfer Protocol
HTML - Hypertext Markup Language
W3C - World Wide Web Consortium
SP - SymposiumPlanner
OA - Organizational Agent
Super-OA - Super Organizational Agent
Sub-OA - Sub Organizational Agent
PA - Personal Agent
EA - External Agent
ESB - Enterprise Service Bus
TD - Top Down
BU - Bottom Up
Prolog - PROgramming in LOGic
OWL - Web Ontology Language
SOAP - Simple Object Access Protocol
IRI - Internationalized Resource Identifier
JMS - Java Message Service
UMO - Universal Message Object
XSL - EXtensible Stylesheet Language
Prova - Prolog+Java
OSGi - Open Services Gateway initiative framework
JVM - Java Virtual Machine
Recent years have seen the introduction of many rule-based multi agent systems (MAS) i.e. JADE and EMERALD. Despite this positive trend in the MAS market, there are no dedicated rule-based agent systems focusing on conference planning. Rule-based agent systems make intelligent decisions based on rule based and ontology driven knowledge bases.

Rule Responder is a tool for creating virtual organizations as multi-agent systems that support collaborative teams on the Semantic Web.

Chapter 1

Introduction

Person-centered and organization-centered profile descriptions using Web 3.0(Semantic Web plus Web 2.0) techniques are becoming increasingly popular. They are often based on the Resource Description Framework (RDF) and include Friend of a Friend (FOAF), Semantically-Interlinked Communities (SIOC), and the ExpertFinder Initiative[34].

Rule Responder is an instantiation of the RuleML-based Rule Responder, which is a tool for creating virtual organizations as multi-agent systems. Distributed users (humans or agents) can interact with the Rule Responder by query-answer conversation protocols. Rule Responder agents process events, queries, and requests according to their rule-based decision and behavioral logic. It can delegate subtasks to other agents, collect partial answers, and send the complete answer(s) back to the requestor. Rule Responder is discussed in this thesis and extended for conference organization applications.
in the form of a case study.

### 1.1 Overview

Query based systems are receiving more attention lately because they can simulate a real world organization by responding to queries from users. A system capable of interacting with Enquiry Users as well as participants can provide an intelligent solution to perform the redundant tasks of an organization.

When studying conference organization systems online, it is obvious that conference organization involves a multitude of procedures, consuming much time and effort of the organizing body. Individual tasks which might not seem laborious at first, bundled together, form a formidable challenge for any Organizing Committee (OC) to tackle. Conference organization typically involves organization partner coordination, sponsoring correspondence, panel participants management, etc. The importance of conference organization systems cannot be overemphasized as they attempt to alleviate the burden of repetitive tasks through the automation of rule-based organization processes. A careful review of the conference management systems shows various systems developed for different purposes.

- Non-Query-Based Conference Organization Systems
- Manuscript Central

Next page...
Manuscript Central \(^1\), developed by ScholarOne, Inc., is the online submission and peer review system used to handle manuscript submissions to journals. This system is currently used by most IEEE and Association of Computing Machinery (ACM) journals. The system is used by authors to upload papers and have them reviewed.

- Microsoft Research Conference Management Tool (MSRCMT)
  Firstly developed for ACM SIGKDD 1999, the MSRCMT \(^2\) is an academic conference management service sponsored by Microsoft Research. Surajit Chaudhury, a Research Area Manager at Microsoft Research is the architect of MSRCMT. Similar to Manuscript Central, the MSRCMT is also a fully-developed system. It is free and hosted by Microsoft Research, but with limited support, since it is developed and managed by a small team.

- EasyChair
  EasyChair \(^3\) is capable of supporting two models: (1) the standard model for conferences having one program committee and (2) the multi-track version for conferences having multiple tracks that have their own program committee.

\(^1\)http://scholarone.com/products/manuscript/
\(^2\)http://cmt.research.microsoft.com/cmt/
\(^3\)http://www.easychair.org/
WitanWeb \(^4\) is a system designed to allow for the online submission, referring and decision making of proposals. The proposals could be papers submitted to a referred publication, grant proposals, award nominations, etc.

- **Query-Based Conference Organization Systems**

  - **SymposiumPlanner-2010**

    After three earlier SymposiumPlanner systems, SymposiumPlanner-2010 has been implemented using a combination of EMERALD \(^5\) and Rule Responder \(^6\). EMERALD is a multi-agent knowledge-based framework, which offers flexibility, reusability and interoperability of behavior between agents, based on Semantic Web and FIPA language standards. The main advantage of this approach is that it provides a safe, generic, and reusable framework for modeling and monitoring agent communication and agreements. Rule Responder is a tool for creating virtual organizations as multi-agent systems that support collaborative teams on the Semantic Web.

  - **SymposiumPlanner-2011**

    After SymposiumPlanner-2010, the 2011 instantiation is implemented using Rule Responder. The SymposiumPlanner-2011 has

---

\(^4\)http://witanweb.ca/cascon2010/WitanWebFAQ.jsp

\(^5\)http://lpis.csd.auth.gr/systems/EMERALDRR/

\(^6\)http://ruleml.org/RuleResponder/
been implemented for organizing two conferences i.e. RuleML-2011@BRF \(^7\) and RuleML-2011@IJCAI \(^8\). Distributed users (humans or agents) can interact with Rule Responder by query answering conversations [CB08] based on an Enterprise Service Bus (ESB). Rule Responder agents will process events, queries and requests according to their rule-based decision and behavioural logic. It can also delegate subtasks to other agents, collect answers, and send the validated answer(s) back to the requester. Since the Rule Responder framework has been conceived, many instantiations of it have been developed in areas as diverse as Service Level Management, Business Process Management, Symposium Planning, and Health Care systems [BP11]. The main benefit from such use is the alleviation of the burden of repetitive tasks through the automation of rule-based organization processes.

Conference organization involves various procedures which make the task difficult both for the users and the developers of such systems. Tasks like responsibility assignment, contact information retrieval, and partner management are complex, and usually involve redundant transactions. One of the foremost challenges of developing systems to perform these tasks is the necessary trade-off between load balancing and speed. A request overload can burden the system and the speed of response of a system to a query can

\(^7\)http://www.businessrulesforum.com/
\(^8\)http://ijcai-11.iiia.csic.es///
make the difference in attracting more organizations to use the system. For this purpose, it is necessary to ensure that the system does not spend too much time on one query, and a powerful enterprise service bus capable of supporting different protocols can help transport messages among disparate knowledge bases without delay and error. Another technical challenge is to provide a user friendly user client, where users can query the system easily without getting confused, and retrieve relevant information.

Query based systems are receiving more attention lately because they can simulate a real world organization by responding to queries from users. A system capable of interacting with Enquiry Users as well as participants can provide an intelligent solution to perform the redundant tasks of an organization. EMERALD depends on external rule base and knowledge base. SymposiumPlanner uses local rule base and external semantic knowledge repositories on the Internet such as DBpedia, Freebase, YAGO, Semantic Web Dog Food to avoid redundancy in knowledge bases. However, SymposiumPlanner -2011 has added complexity to cater to requirements of the multiple events related to RuleML Symposia. This effects system performance as well as making it harder to maintain. The system also suffers from some faults in design which can be fixed to help it perform to its full potential.
1.2 Thesis Objective and Methodology

Earlier conference support systems include SymposiumPlanner-2007 through -2011 series of Rule Responder instantiations9, which focus on question answering as part of the official websites of the RuleML Symposia. The SymposiumPlanner instantiations span various implementations from initial versions in 2007, 2008, and 2009, to the SymposiumPlanner-2010 (in EMERALD and Rule Responder) instantiation based on EMERALD (hence Jade). This thesis will design and implement, in Rule Responder, a distributed query system supporting symposium Organization Committees (OCs). Our improved SymposiumPlanner design and implementation will be explored in a use case, the RuleML-2012 Symposium. Its support will build on the earlier SymposiumPlanners and an extensive literature survey, distilling design principles and implementation techniques for future SymposiumPlanners from the lessons learned so far, as started in. Possible OC support includes:

- Answering question by (potential) participants about the conference such as important dates, paper topics, chairs, program schedule, etc.

- Coordinating chair responsibilities.

- Helping the program chair to monitor and if necessary move important dates.

- Helping the publicity chair with sponsoring correspondence

9http://ruleml.org/SymposiumPlanner
Our overarching objective is to design, implement and evaluate a distributed query system supporting symposium organization committees. This is achieved by the following concrete objectives.

1. In our system the agent communication will be achieved in a hierarchical way, the agent-to-agent communication is channelled through the Organizational Agent. In particular, when an External Agent asks a question to the organization, it does not know any personal agent such as the one that ultimately might answer the question. The query will be forwarded to the Organizational agent which will then delegate it to an appropriate Personal Agent.

2. Our system will provide Personal Agents which can assist the local entities of a virtual organization. Often these are humans but it can also be services or applications. The Personal Agents are semi-autonomous in nature as we do not think it is possible to replace an actual human in the organization, however they can provide a helping hand in dealing with certain tasks that have a certain degree of redundancy in terms of information.

3. Our system will benefit from the Personal Agent knowledge bases, building on and modifying the encoded knowledge. This opportunity can be utilized to pre-structure the knowledge bases for the Personal Agents.

4. Our system will be able to be deployed on a local machine as well
as the Internet. The Symposium planning websites online can host the system to assist users find information by issuing queries to the system. The system is developed locally and hence can be deployed in a local environment as well depending on needs of an organization. However, since our implementation is mostly dealing with Symposium Organizing bodies, it is more useful online.

5. Our system will support the Questions&Answers (Q&A) parts of the official websites for the RuleML Symposia. It will embody responsibility assignment, automated first-level contacts for information regarding the symposium, helping as well in sponsoring correspondence.

6. Our system will distill design principles and implement techniques for SymposiumPlanner-2012 from the lessons learnt so far.

7. Our system will integrate existing factual information on the Internet, this will help us avoid redundancy in the knowledge bases of the SymposiumPlanner agents.

8. Our system will reunify the business logic into one Organizational Agent instead of three in the previous configuration of 2011.

9. Our system will involve development of virtual RuleML 2012 OC. This implementation will also help in evaluating our system, as we will solicit feedback from the real world OC who it is designed to assist. This practise can result in continuous revisions of the knowledge bases of
the Personal Agents, the Organizational Agent and perhaps even the system architecture. These improvements can provide the ground for further improvements in future designs and implementations of the SymposiumPlanner.

1.3 Thesis Organization

The organization of the thesis is as follows. Conference management and other related basic concepts in this thesis are introduced in Chapter 2. In Chapter 3, we describe Rules and Rule engines. Then in Chapter 4, we discuss the SymposiumPlanner as a Rule-based system with its personal profiles and global knowledge bases as well as the coordination among them. We discuss the various architectural design principles along with the Rule Responder architecture on which our system is based on. In Chapter 6, we discuss the SymposiumPlanner 2012 as a query based conference management system with details of its architecture as well as the communication middleware driving the interaction of the various parts of the system. Finally, we discuss our conclusions with an overview of contributions and future work are in Chapter 7.
2.1 The Semantic Web

Tim Berner Lee put forward the concept of Semantic Web. In his own words (http://www.w3.org/RDF/Metalog/docs/sw-easy), "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

W3C defines Semantic Web as "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of re-searchers and industrial partners. It is based on the Resource Description Frame-work (RDF), which integrates a variety of applications using XML for syntax and URIs for naming."
2.2 Metadata and Resource Description Framework

In this section, we introduce two important notions in the Semantic Web: metadata and Resource Description Framework (RDF).

2.2.1 Metadata

Metadata is defined as structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource” in [1]. W3C further elucidates the concept of metadata by alluding to its capability of being machine readable information. Metadata can benefit
resource discovery, interoperability, digital identification as well as archiving and preservation.

2.2.2 Resource Description Framework

The Resource Description Framework (RDF), recommended by W3C, is a framework for modeling information at a metadata level of the Web [62]. RDF is designed for widespread, decentralized use.” [30]. RDF is written in XML. RDF uses a triple expression, consisting of a subject, a predicate and an object. The subject represents the resource; the predicate represents the relationship between the subject and the object while the object is the object in this relationship.

2.2.3 RDF Schema (RDFS)

"RDF Schema is a language for describing vocabularies in RDF”. RDF provides a mechanism with the purpose of integrating multiple metadata schemas extracted from distributed information [11].

2.3 Intelligent Agents

An agent is an encapsulated computer system made up of an architecture and a program. This computer system should have:

- Be situated in some environment.
- Be able to perceive its environment.

- Be capable of autonomous action within that environment.

- Have some kind of design objectives.

### 2.3.1 Agent communication

For the formation of an intelligent system if there isn’t some kind of communication to share knowledge and experiences [17]. A fundamental characteristic of multi-agent systems is that individual agents communicate and interact with one and another. This communication is accomplished through the exchange of messages and, to understand each other. However, any such arrangement is made possible if the agents are able to agree on the format and semantics of these messages (http://www.iro.umontreal.ca/ vaucher/A-gents/Jade/primer4.html).

### 2.3.2 FIPA

Foundation for Intelligent Physical Agents (FIPA)\(^1\) has carefully built standards and recommendations for use in multi-agent systems. This allows agents to have some sort of lingua franca.

\(^1\)http://www.fipa.org/
2.3.3 Agents on the Semantic Web

We can accomplish a number of tasks with agents and the Semantic Web [16]. For example:

- An agent turns documents into formal Semantic Web-based knowledge.
- A multi-agent system that can operate in its community to build and maintain Linked Data sets.

2.4 Conference Management

A conference management system is web-based software that supports the organization of scientific conferences. It helps the program chair(s), the conference organizers, the authors and the reviewers in their respective activities (http://en.wikipedia.org/wiki/Conference_management_system). Conference management typically involves submission reviewing, organization partner coordination, sponsoring correspondence, panel participants management, etc. Some of these tasks can be (partially) automated. Possible OC support includes:

- Answering question by (potential) participants about the conference such as important dates, paper topics, chairs, program schedule, etc.
- Submission reviewing.
- Coordinating chair responsibilities.
- Helping the program chair to monitor and if necessary move important dates.

- Helping the publicity chair with sponsoring correspondence.
Chapter 3

Rule Languages and Tools

3.1 Rules

Rules are central to knowledge representation for the Semantic Web [8]. The sections in this chapter describe the rule languages relevant to this thesis. Rule markup languages are the vehicle for using rules on the Web and in other distributed systems. They allow publishing, deploying, executing and communicating rules in a network. They also enable exchange of information between systems and tools. A rule markup language is a concrete syntax for the Web[23].
3.2 XSLT

XSLT (Extensible Stylesheet Language Transformations) is a declarative, XML-based language used for the transformation of XML documents. The original document is not changed; rather, a new document is created based on the content of an existing one. (http://en.wikipedia.org/wiki/XSLT)

XSLT consists of three parts:

- XSL Transformations (XSLT): A language for transforming XML.

- XML Path Language (XPath): An expression language used by XSLT to access parts of an XML document.

- XSL Formatting Objects (XSL-FO): An XML vocabulary for specifying formatting semantics.

XSLT is a language of interest in this thesis. XSLT is used for transforming XML documents into other XML documents. XSL specifies the style of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting XML vocabulary XSL-FO [19].

A transformation in XSLT is expressed as rules for transforming a source tree into a resultant tree. The transformation is achieved through associating patterns with the template. Each template contains instructions for transformation in the form of patterns.

In Rule Responder, XSLT is used to transform XML documents (RuleML)
When a user sends a RuleML query through the External Agent (EA) to the OA, the query needs to be transformed for Organizational Agent’s comprehension (OA processes Prova rules). In this case XSLT is used to make this communication between the External and Organizational Agent possible. XSLT is also used to transform(translate) messages among the agents(i.e. OA and PA).

3.3 Rule Languages

3.3.1 RuleML

The Rule Markup Language (RuleML, www.ruleml.org) is a markup language developed to express a family of Web rules in XML for deduction, rewriting, and reaction, as well as further inferential, tranformational, and behaviral tasks. (Rule Markup Languages and Semantic Web Rule Languages)

3.3.2 POSL

Prolog is the acronym of Programming in Logic. It is based on the mathematical notions of relations and logical inference, and therefore a logic programming language using Horn clauses (http://www.cs.wwc.edu/KU/PR/Prolog.html).

POSL (Positional-Slotted Language), derived from Prolog, is a human-readable format for Semantic Web Knowledge that combines Prolog’s positional and
F-logic’s (Frame Logic, which logic basis of frame-based and object-oriented languages for data and knowledge representation [cite]) slotted syntaxes for representing knowledge (comprising rules and facts) in the Semantic Web [20]. Harold Boley in [7] describes that POSL not only contains many kinds of assertional-logic and object-centered modeling styles, but also achieves conciseness as well as orthogonality at large.

The compactness of POSL helps humans read and write it easier than XML. The capability of POSL to be interconvertible with RuleML makes it even more attractive as a machine readable language like XML [3].

3.4 Semantic Web Rule Languages, Rule Engines and Tools

In contrast to the XML-based rule markup languages, the Semantic Web Rule Languages are human-readable. Typically they are designed as “compact” presentation languages for human interpretation. Despite being easier for the human mind, these languages can be used for interchange purposes through the use of rule engines.

3.4.1 Prova

Prova(http://www.prova.ws/) is both a Web rule language as well as a highly expressive distributed (Semantic) Web rule engine which supports
complex reaction rule-based workflows, rule-based complex event processing, distributed inference services, rule interchange, rule-based decision logic, dynamic access to external data sources, Web Services and Java APIs [24].

One of the key advantages of Prova is its separation of logic, data access, and computation as well as its tight integration of Java, Semantic Web technologies and enterprise service-oriented computing and complex event processing technologies.

The Prova rule engine supports different rule types:[10]

- Derivation rules to describe the agent’s decision logic.
- Integrity rules to describe constraints and potential conflicts.
- Normative rules to represent the agent’s permissions, prohibitions and obligation policies.
- Defeasible rules to prioritize rules for, e.g. handling conflicts between agent’s goals and modularization of the agent’s KB to support multiple roles of an agent.
- Global ECA-style reaction rules to define global reaction logic which are triggered on the basis of detected (complex) events.
- Messaging reaction rules to define the agents conversation-based workflow reactions and behavioral logics based on complex event event processing.
General Prova Features

The Prova language consists of four main components [22]:

- Simple atomic terms: These can be a constant or a variable.

- Compound terms (lists): They are represented generically in the Prolog syntax as \([\text{Head—Rest}]\), and are used for agent communication.

- Facts: They are records with their elements being any term (a constant, variable, or (possibly recursive) list), such as

\[
\text{predicate}(\text{arg}_1,\ldots,\text{arg}_n).
\]

- Rules: They are Horn rules which are written in the standard Prolog-like way:

\[
\text{headLiteral(} \text{args}_h) \quad \text{:-}
\text{bodyLiteral}_1(\text{args}_1),\ldots,\text{bodyLiteral}_n(\text{args}_n).
\]

Here \(\text{:-}\) is pronounced as IF.

Fact: If a rule has no body.

Goal: If a rule has no head.

The following Prova is coded in the OA to match a received query against some existing interfaces (facts). If no match is found, SymposiumPlanner reacts with a "no interface found" message and the rule fails.
processMessage(XID, From, Primitive, [X|Args]) :-
    not(interface([X|Args], ModeDeclarations, Description)),
    sendMsg(XID, esb, From, "answer", noPublicInterface
    (interface ([X|Args]))),
    sendMsg(XID, esb, From, "no_further_answers", [X|Args])),
    fail().

- ProcessMessage(): This rule is for processing the user's query. It has
the following parameters.
(a) XID - Name of OA.
(b) From is the name of the endpoint.
(c) Primitive is the user name (e.g., User).
(d) X is the relation name surrounded by angle brackets as in ¡Rel¡,
(e) Args is the arguments of the relation.

- Interface: This describes the valid format for the user query which is
acceptable for the OA.
If no match is found among interfaces in the OA. Then, the system
responds with a “no interface found” message.
In this case, the OA also sends another message with the performative
”no further answers” so that the system can
terminate the search rather than wait indefinitely for more answers.

Reactive Messaging
In Prova, reactive messaging is used for organizing distributed Prova engines into a network of communicating agents. A Prova agent contains message passing primitives and communicates using protocols. However, in our system the distribution aspect of Prova agent is not part of the Prova-agent rulebase code because the ESB is used for communication.

A Prova-agent rulebase, employs pattern matching for message processing. The following parameters are used for Prova message passing primitives in both sending and retrieval of messages.

- **XID** - The Conversation ID of the message.
- **Protocol** - Protocol being used for messaging.
- **Destination** - The receiver on the sending end; and the sender on the receiving end.
- **Performative** - The message type characterizing the meaning of the message, such as an answer.
- **Payload** - A Prova list containing the actual content of the message.

The user specifies all those parameters, which should be constants, and leaves those that require further reasoning as free variables.

### Sending Messages

Message sending is initiated when the Prova engine processes the literal "sendMsg" which is a premise of any rule. Let us now discuss the arguments of the built-in sendMsg().
Conversation ID

The Conversation ID is used for keeping track of a message so that the request and response related to that conversation are contextualized into a single conversation. In SymposiumPlanner, this parameter is a constant value; hence, the send message is a follow-up to an on-going conversation. Let us look at some example code to get a better understanding of the typical follow-up pattern which part of the of SymposiumPlanner's processMessage() rule:

```plaintext
sendMsg(XID,esb,Agent, "query", contactDocSymposiumChair
(Meeting, Chair, FirstName, LastName, Title,
 Email, Telephone)),
println(["Sent message to the ",Agent," PA."]),
println([""]),
rcvMult(XID,esb,Agent,"answer", substitutions(Meeting,
 Chair, FirstName, LastName, Title, Email, Telephone)),
```

In the above set of messages, the SymposiumPlanner OA uses sendMsg() to send the user's query "contactDocSymposiumChair" to the Agent. Then some messages are printed to show the progress in processing this request. Next, some answers are received from the Agent using rcvMult(). We must note that both sendMsg() and rcvMult() have the same conversation ID (XID).

Protocol (Next page)
In SymposiumPlanner, we have two types of protocols.

- Async Protocol: This protocol is for limiting the processing to a unique thread depending on the conversation ID.

- ESB Protocol: This protocol is for designating the agent as the forwarder for dispatching the message.

In SymposiumPlanner, the Mule ESB components are used as a container for Prova agents. Thus, if the protocol is set to esb and the destinate is set to the logical endpoint on the Mule ESB, the message can be delivered by Mule to the agent.

When a Mule component receives a message from outside, the inbound protocol name is swapped from esb to async so that the asynchronous protocol is used.

**Destination**

In SymposiumPlanner, the destinations are the agent names. In case of OA/PA conversations. The SymposiumPlanner has to utilize a assigned() to find out the responsible agent capable of answering a query based on its role in the Role Assignment Matrix.

When assigned() starts processing, the sub-rule import() retrieves the address of the ontology, which includes the PA roles and responsibilities for a particular instance. Next the subrule reasoner() retrieves the rule language in which the OWL ontology is implemented (in our case OWL). Then the built-in rdf() is used for reasoning in the OWL ontology and retrieving the...
responsible agent’s name and its role based on the responsibility assigned to it in the Role Assignment Matrix.

When Prova obtains the name of the PA, the destination is the logical name of the PA’s endpoint. Since, the Mule ESB is used for message routing, the logical name of the PA’s endpoint is configured via the Mule configuration file (i.e. mule-config).

Here, one PA of the Sub-Organizational Agent representing RuleML 2012@ ECAI is configured inside the Mule configuration as a topic.

```xml
<http:.endpoint name="ruleml2012ATecai_PublicityChair" address="${TOMCAT}/${ECAI2012OName}${ECAI2012Topic1}/"/>
```

**Performatives**

Performatives describe the pragmatic envelope for the message content. Prova agents allow the developers to encode performative based reasoning. Processing messages with the answer performative involves a receiving and a sending process. In the following example, the OA receives answers from the PA representing the PublicityChair and sends them to the External Agent.

```java
rcvMult(XID,esb,Agent,"answer",
    substitutions(Meeting, Chair, FirstName,
    LastName, Title, Email, Telephone)),
println(["Received message from the ",Agent," PA."]),
sendMsg(XID,esb,From, "answer", contactPublicityChair
    (Meeting, Chair, FirstName, LastName, Title, Email, Telephone)),
```

27
Each time the rcvMult() receives an answer from the PA, the sendMsg() sends the answer to the browser (Web Page), till no more answers are left. The println() is used to print the status of message handling. The rcvMult() waits indefinitely for answers; in order to make sure this doesn’t happen a timeout on waiting for an answer or a termination message are used. This termination message can also be issued in case there is no communication between the OA, PA and the EA. Since, we have explicitly set the communication timeout to be around 3 minutes. After this time, the termination is carried out if there is no communication. The termination message is also used when the PA sends a ”no further answers” response so that the OA does not wait for an answer indefinitely.
Payload

Payload is the main content of the message that makes a query. In Prova, the content of the inbound query must be a ProvaList. Each element in a ProvaList is either a Prova constant, variable or another list as shown here for illustrative purposes,

\[
\text{rcvMsg}(XID, esb, From, \text{Performative}, [X|Args]) \leftarrow \\
\text{understandPerformative}(XID, From, \text{Performative}, [X|Args]), \\
\text{processMessage}(XID, From, \text{Performative}, [X|Args]).
\]

In the list, X— Args, the X and the Args can be any term (e.g., a list, a variable or even a constant). These terms can be repeated as many times as the number of terms in the user query.

Methodology of Message Receiving

Prova uses Reaction rules for sending and receiving messages. The Reaction rules look precisely like Horn rules or Prova rules however, their semantics (primitives) are aligned with Reaction rules rather than derivation rules. SymposiumPlanner uses the following two types of Reaction rules:

Receiving Messages

Prova uses reaction rules for sending and receiving messages. The Reaction rules look precisely like Horn rules or Prova rules (with the predicate symbol
"rcvMsg" at the head) but their semantics (primitives) are more along the lines of reaction rules rather than derivation rules. SymposiumPlanner uses the following two types of reaction rules:

**Global Reaction Rules**

The simplest form of a reactive rule is one when the head of the rule is a 
message receiving primitive distinguished by the *rcvMsg* predicate symbol. (prova.ws/Using Reaction rules)

\[
\text{rcvMsg}(\text{XID}, \text{esb}, \text{From}, \text{Performative}, [X|Args]) :-
\]

\[
\text{understandPerformative}(\text{XID}, \text{From}, \text{Performative}, [X|Args]).
\]

\[
\text{processMessage}(\text{XID}, \text{From}, \text{Performative}, [X|Args]).
\]

When *rcvMsg* is executed, the function *understandPerformative()* interprets the "Performative" to find out how to process the payload i.e \([X—Args]\). The *rcvMsg* rule has three parameters for accepting messages from a sender, which are:

- XID
- Protocol
- From

The rule has a rulebase lifetime scope, i.e. it is active while the rulebase runs in a Prova engine. The global scope means that the rule is ready to receive any number of messages as they arrive to the agent. This rule in simple
terms waits for messages that match the pattern specified in brackets after \texttt{rcvMsg} and responds to such matching messages with logic reasoning contained in the body of the rule.

**Inline Reaction Rules**

The inline reaction rules are more dynamic and volatile. Their scope can be changed from accepting just one message to a specified number of messages. They can also be limited by a timeout. These reaction rules are of special use in workflow and event processing.

The fundamental idea behind inline reaction rules is comparable to closures or continuations. The reaction is created as part of evaluating the body of a rule when a message receiving primitive \texttt{rcvMsg} is part of that body. The OA is able to understand that the Agent has no more answers when the performative in the \texttt{rcvMsg} is "no further answers".

\begin{verbatim}
rcvMsg(XID, esb, From, Performative, [X|Args]),
understandPerformative(XID, From, Performative, [X|Args]),
rcvMsg(XID,esb,Agent, no_further_answers, Payload),
println(['-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-'])
println([' COMPLETE '])
println(['-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-'])
sendMsg(XID, esb, From, no_further_answers, [X|Args]).
\end{verbatim}

The premises after the "\texttt{rcvMsg}" premise wait till "\texttt{rcvMsg}" is matched. This ends up creating a closure that contains all the remaining literals in the body.
of the rule along with the dynamically generated reaction that is waiting for
the pattern specified in brackets after rcvMsg. The conversation ID "XID"
is used for correlation. Hence, including "rcvMsg" in the body any Prova
rule results in freezing the current state of all the context and boy literals
following rcvMsg.

The rcvMsg works only once, if the reaction rule is to be indefinitely, then
rcvMult is used. In SymposiumPlanner, Prova is used for representing the
shared knowledge of Organizational Agent (OA) for organizing the workflow
of messages among agents representing different members of a conference or
an organization.

The OA uses rules to delegate queries to and from the External Agent (EA)
as well as the Personal agents (PA). For this purpose it uses the responsi-
bility assignment matrix for assigning tasks to the responsible agents. After
receiving answers from the PAs, the OA is able to send them back to the EA
(human user or a service).

3.4.2 Reaction RuleML

Reaction RuleML is a general, practical, compact and user-friendly XML-
serialized language and rule interchange format for the family of reaction
rules. It incorporates different kinds of production, action, reaction, complex
event notification / messaging and KR temporal/event/action logic rules into
the native RuleML syntax using a system of step-wise extensions.

Reaction RuleML provides several layers of expressiveness for adequately
representing an agent’s logic for interchanging events (queries, actions, event data) and rules. [28]

The building blocks of Reaction RuleML are [27]:

- One general (reaction) rule form “Rule” that can be specialized to e.g. production rules, trigger rules, ECA rules, messaging rules ....

- Three execution styles defined by the attribute @style

  - Active: ‘actively’ polls/detects occurred events in global ECA style.

  - Messaging: waits for incoming complex event message and sends outbound messages as actions.

  - Reasoning: Knowledge representation derivation and event/action logic reasoning and transitions.

- Messages “Message” define inbound or outbound event message.

The general syntax of a reaction rule consists of six partially optional parts as shown in fig 3.1.

Inbound and outbound messages are used to interchange events and rule bases between agents.

3.4.3  **OO jDREW**

OO jDREW, stands for Object Oriented java Deductive Reasoning Engine for the Web. It is an object-oriented extension of jDREW (java Deductive
<Rule style="active" evaluation="strong">
  <label> <!-- metadata --> </label>
  <scope> <!-- scope --> </scope>
  <qualification> <!-- qualifications --> </qualification>
  <oid> <!-- object identifier --> </oid>
  <on> <!-- event --> </on>
  <if> <!-- condition --> </if>
  <then> <!-- conclusion --> </then>
  <do> <!-- action --> </do>
  <after> <!-- postcondition --> </after>
  <else> <!-- else conclusion --> </else>
  <elseDo> <!-- else/alternative action --> </elseDo>
  <elseAfter> <!-- else postcondition --> </elseAfter>
</Rule>

Figure 3.1: Reaction Rule Syntax

<Message mode="outbound" directive="pragmatic performative">
  <oid> <!-- conversation ID --> </oid>
  <protocol> <!-- transport protocol --> </protocol>
  <sender> <!-- sender agent/service --> </sender>
  <content> <!-- message payload --> </content>
</Message>

Figure 3.2: Reaction Rule Message
Reasoning Engine for the Web). OO jDREW is a Java reasoning engine for executing RuleML [4].

The internal agents of SymposiumPlanner are “currently” implemented using the reasoning engines Prova and OO jDREW. OO jDREW is used for certain personal agents (PAs) while Prova is used for other PAs as well as the organizational agent.

OO jDREW has two main modes of operation top-down and bottom-up [3].

- Bottom-Up Execution: It is used to infer all derivable knowledge from a set of clauses (forward reasoning).

- Top-Down Execution: It is used to solve a query on the knowledge base (backward reasoning).

SymposiumPlanner primarily uses top-down execution due to the nature of the system as in query-answering services which follows a sequence of steps as follows:

- The query-answering service initiates with a query that is sent by the EA to an organizational agent.

- To obtain a query for processing, the OO jDREW engine first parses the message and translates the message which is a RuleML query into OO jDREW.

- OO jDREW loads that personal agent’s FOAF profile (stored as POSL) and based on rules defined in the profile, derives the answer.
- The derived answer is sent back to the EA as a Reaction RuleML event message.
- If the FOAF profile requires a user defined taxonomy then an RDF Schema file of user classes is parsed and the query is parsed.
- OO jDREW runs the query against the knowledge base.
- Solutions are derived.
- Solutions are serialized into Reaction RuleML.
- Solutions are sent back to the requesting agent.

Each message contains a performative wrapper that is used by Symposium-Planner to understand how to interpret and use the message. The information in the message contains the following:

- The sender of the message.
- The transport protocol used by the ESB.
- OID: Conversation id of the current message.
- Content: (Incoming message queries that are to be answered) (Outgoing message query results).
- Mode: This indicates whether a message is outbound or inbound.
- Directive: This distinguishes whether a message content is query or an answer.
Let’s look at an example of the query-answering service, we want to know who should be contacted about the symposium’s general chair?

When the message is received by the OA, it can decide which agent is capable of answering this query. Once the query is successfully executed, the OO jDREW will send the message to the RuleML-2012-ECAI organizational agent. The organizational agent will parse the subquery solution and send one final message back to the external agent.
<table>
<thead>
<tr>
<th>Incoming Message to QA from EA</th>
<th>Incoming Message (sub query) to OO jDREW from QA</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;RuleML&gt;</code></td>
<td><code>&lt;content&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Message mode=&quot;outbound&quot;</code></td>
<td><code>&lt;Atom&gt;</code></td>
</tr>
<tr>
<td><code>directive=&quot;query-sync&quot;</code></td>
<td><code>&lt;Rel&gt;person&lt;/Rel&gt;</code></td>
</tr>
<tr>
<td><code>&lt;oid&gt;</code></td>
<td><code>&lt;Var&gt;Name&lt;/Var&gt;</code></td>
</tr>
<tr>
<td><code>SymposiumPlannerSystem&lt;/Ind&gt;</code></td>
<td><code>&lt;Var&gt;Role&lt;/Var&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Protocol&gt;</code></td>
<td><code>&lt;Var&gt;Title&lt;/Var&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Sender&gt;</code></td>
<td><code>&lt;Var&gt;Email&lt;/Var&gt;</code></td>
</tr>
<tr>
<td><code>User&lt;/Ind&gt;</code></td>
<td><code>&lt;Var&gt;Telephone&lt;/Var&gt;</code></td>
</tr>
<tr>
<td><code>&lt;content&gt;</code></td>
<td><code>&lt;Atom&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Rel&gt;getContact&lt;/Rel&gt;</code></td>
<td><code>&lt;Atom&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Ind&gt;ruleml2012ATeCAI_GeneralChair&lt;/Ind&gt;</code></td>
<td><code>&lt;Var&gt;Contact&lt;/Var&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Atom&gt;</code></td>
<td><code>&lt;Var&gt;Contact&lt;/Var&gt;</code></td>
</tr>
<tr>
<td><code>&lt;content&gt;</code></td>
<td><code>&lt;Atom&gt;</code></td>
</tr>
<tr>
<td><strong>Outgoing Message from QA to EA</strong></td>
<td><strong>Outgoing Message from QA to EA</strong></td>
</tr>
<tr>
<td><code>&lt;content&gt;</code></td>
<td><code>&lt;Atom&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Rel&gt;person&lt;/Rel&gt;</code></td>
<td><code>&lt;Rel&gt;getContact&lt;/Rel&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Ind&gt;GuidoGovernatori&lt;/Ind&gt;</code></td>
<td><code>&lt;Ind&gt;ruleml2012ATeCAI_GeneralChair&lt;/Ind&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Ind&gt;GeneralChair&lt;/Ind&gt;</code></td>
<td><code>&lt;Ind&gt;update&lt;/Ind&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Ind&gt;PHD&lt;/Ind&gt;</code></td>
<td><code>&lt;Expr&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Ind&gt;guido@email.com&lt;/Ind&gt;</code></td>
<td><code>&lt;Fun&gt;person&lt;/Fun&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Ind&gt;1-800-800-800&lt;/Ind&gt;</code></td>
<td><code>&lt;Ind&gt;GuidoGovernatori&lt;/Ind&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Atom&gt;</code></td>
<td><code>&lt;Ind&gt;PHD&lt;/Ind&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Atom&gt;</code></td>
<td><code>&lt;Ind&gt;guido@email.com&lt;/Ind&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Atom&gt;</code></td>
<td><code>&lt;Ind&gt;1-800-800-800&lt;/Ind&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Atom&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

The sub query corresponds to the head of a rule in (POSL), executed in OO jDREW. Following is a sample rule for a full FOAF profile,

\[
\text{person}(\text{?Person, ?Role, ?Title, ?Email, ?Telephone}) \quad \text{:-} \\
\quad \text{contact}(\text{?Person, ?Email, ?Telephone}), \\
\quad \text{role}(\text{?Person, ?Role}), \\
\quad \text{title}(\text{?Person, ?Title}).
\]
In this chapter, we describe how SymposiumPlanner is implemented as a Rule Responder instantiation, as well as its architecture and functionality. Rule Responder is a service-oriented middleware tool that can be used by virtual organizations for automated rule-based collaboration [29]. Distributed users (humans or agents) can interact with Rule Responder by query answering conversations [15] based on an Enterprise Service Bus (ESB). Rule Responder’s architecture realizes a system of personal agents (PAs) and organizational agents (OAs), accessed by external agents (EAs), on top of an Enterprise Service Bus (ESB) communication middleware [15]. The rule-based PAs represent, as their ‘dynamic profiles’, all of the participating human members of the virtual organization modeled by the Rule Responder. An OA constitutes an intelligent query dispatching system, using a rule engine execution environment for selectively delegating queries to one of the
4.1 Approaches to Knowledge Representation

In Rule Responder we make use of two paradigms of knowledge representation: Ontologies and rules. The OA uses a Responsibility Assignment Matrix to delegate incoming queries to particular PAs based on their responsibilities. This is achieved through the use of an ontology (OWL Lite). This Responsibility Assignment Matrix can be used by the Rule Responder agent via querying it with the Semantic Web built-ins of Prova, binding the respective roles and responsibilities to typed variables in the agent’s rule logic.

The Organizational Agent is implemented using the Prova Semantic Web rule engine. The Organizational Agent uses reaction rules that describe its policies, regulations, opportunities, etc. Each Personal Agent uses derivation rules to answer queries relevant to the supported chair’s role. Hence, our framework uses a combination of rules and ontologies.

4.2 Distributed Architecture Approaches

Distributed systems are fast gaining importance in Computer Science. While the Semantic Web improves the understanding of Web-based information, the combination of the Semantic Web and distributed systems give rise to a distributed rule based system. A careful study of distributed system archi-

PAs.

4.1 Approaches to Knowledge Representation

In Rule Responder we make use of two paradigms of knowledge representation: Ontologies and rules. The OA uses a Responsibility Assignment Matrix to delegate incoming queries to particular PAs based on their responsibilities. This is achieved through the use of an ontology (OWL Lite). This Responsibility Assignment Matrix can be used by the Rule Responder agent via querying it with the Semantic Web built-ins of Prova, binding the respective roles and responsibilities to typed variables in the agent’s rule logic.

The Organizational Agent is implemented using the Prova Semantic Web rule engine. The Organizational Agent uses reaction rules that describe its policies, regulations, opportunities, etc. Each Personal Agent uses derivation rules to answer queries relevant to the supported chair’s role. Hence, our framework uses a combination of rules and ontologies.

4.2 Distributed Architecture Approaches

Distributed systems are fast gaining importance in Computer Science. While the Semantic Web improves the understanding of Web-based information, the combination of the Semantic Web and distributed systems give rise to a distributed rule based system. A careful study of distributed system archi-
4.2.1 Distributed System

A distributed system is a set of computer processes that appear to the user as a single system. The components of a distributed system are connected through a topology. Our decision on choosing our topology was based on studying the pros and cons of the different topologies in use. A discussion on topologies follows.

4.2.2 Hierarchical (Star Topology, Client-Server-Like Architecture)

A star topology (hierarchy) constitutes simple but effective distributed topology. The star topology connects all nodes through a centralized hub. This is also known as client-server-like architecture where the server is the hub and all the clients are outside spokes. Even if a spoke is broken then it will not affect the rest of the spokes.

4.2.3 Networked (Full/Partial Mesh Topology, Peer-to-Peer-Like Architecture)

The full mesh topology is a fully redundant topology, meaning that any node can send data to any other node and this ensures that there is always information flow to any node if a node fails [14]. However, there is a problem
with this arrangement. It requires many connections and is not a pragmatic approach when a large number of devices are connected through the topology.

The alternative in that case is a partial connected mesh topology which ensures information flow from one node to any other. If a large number of devices are connected in a topology then partial mesh topology becomes more efficient than a star topology.

In our SymposiumPlanner, the star topology connects the specialized PAs ‘spokes’ to the centralized OA ‘hub’. A star topology is partially fault tolerant: If a spoke is broken then it will not affect the rest of the spokes \(^1\). However, if the hub is broken then the system ceases to function.

\(^1\)http://learn-networking.com/network-design/a-guide-to-network-topology
Figure 4.2: Mesh Topology

Figure 4.3: Partial Mesh Topology
4.3 Rule Responder Architecture

The Rule Responder architecture realizes a system of personal agents (PAs) and organizational agents (OAs), accessed by external agents (EAs), on top of an Enterprise Service Bus (ESB) communication middleware [25]. The semi-autonomous PAs and OAs are implemented by a rule engine each which acts as an inference and execution environment for the rule-based decision and behavioral logic of that agent (Principles of SymposiumPlanner). The rule-based PAs represent as their 'dynamic profiles', all the participating human members of the virtual organization modelled by the SymposiumPlanner.

An OA constitutes an intelligent filtering and dispatching system using a rule engine environment for either blocking incoming queries or selectively delegating them to other agents. The communication middleware implements an Enterprise Service Bus (ESB) supporting various transmission protocols (e.g. JMS, HTTP, SOAP). The External Agents (EAs) can interact with the virtual organization via its public communication interface (e.g. an HTTP endpoint interface to an OA as the “single point of entry”. The SymposiumPlanner uses a Web browser for human-machine interaction.

The Rule Responder blends and tightly combines the concepts of multi-agent systems, distributed rule management systems, as well as service-oriented and event driven architectures. The agent-to-agent communication must channel through the Organizational Agent (OA). The External Agent is unaware of the capabilities of the Personal Agents in terms of answering queries.
It must communicate with the OA which is able to evaluate the incoming query and delegate it to the PA best able to respond to that query.

4.4 Communication Middleware

A lot of the early ESB products had a history in the enterprise application integration (EAI) market. We can identify two major differences between enterprise service bus (ESB) and EAI products. The first is the change from the hub-and-spoke model in EAI products to a bus-based model in ESB products. The hub-and-spoke model is a centralized architecture, where all data exchange is processed by a hub [12]. The bus model uses a distributed architecture, in which the ESB functionality can be implemented by several physically separated functions. A second difference between EAI and ESB products is the use of open standards. To seamlessly handle message-based interactions between the agents/services and other agents/services using disparate complex event processing (CEP) technologies, transports, and protocols, the Mule open-source ESB\(^2\) is used in our system as the communication middleware. This ESB allows deploying the rule-based agents as highly distributed rule inference services installed as Web-based endpoints on the Mule object broker and supports the communication in this rule-based agent processing network via a multitude of transport protocols. That is, the ESB provides a highly scalable and flexible application messaging framework to

\(^2\)http://www.mulesoft.org/
Rule Responder Legend:

RR: Rule Responder
VO: Virtual Organization
EA: External Agent
OA: Organizational Agent
PA: Personal Agent
CA: Computing Agent
EU: Enquiry User
PU: Profile User
KB: Knowledge Base

HTTP Request
→ Creates

Note that the PAs shown could themselves be internally structured as VOs and so on, to any depth of recursion, since the entire RR architecture can be considered holonic system (cf = http://cs.unb.ca/~boley/papers/RuleResponderAgents.pdf)

Figure 4.4: Rule Responder Architecture
communicate synchronously or asynchronously amongst the ESB-local agents and with agents/services on the Web.

Mule open-source ESB allows the deployment of rule-based agents on the Mule object broker and supports the communication in this rule-based agent processing network via a multitude of transport protocols. Several agent services which at their core run a rule engine are installed as Mule components which listen at configured endpoints, e.g., JMS message endpoints, HTTP ports, SOAP server/client addresses or JDBC database interface. The large variety of transport protocols provided by Mule can be used to transport the messages to the registered endpoints or external applications/tools. Usually, JMS is used for the internal communication between distributed agent instances, while HTTP and SOAP is used to access external Web services. The use of Mule allows architectural flexibility by decoupling the functional components of Rule Responder from the communication components Chappell04.

A discussion on the Mule ESB follows.
5.1 Interaction Between Organizational and Personal Agents

The Organizational agents are employed to describe the organization as a whole. An OA contains a knowledge base that describes the organization’s policies, regulations, and opportunities. The OA knowledge base contains condition/action/event rules as well as derivation rules. [34]

When a RuleML-formalized version of a query is received by an OA, it has to decide who the correct person is to contact for this query. When the correct person has been selected by the OA. The OA delegates the query to that Personal Agent (PA). The PA will then respond with the correct information
In the development of the SymposiumPlanner-2012 use case we will be taking a proactive approach. We are designing, implementing, and deploying the system well in advance to the actual August 2012 symposium collocated with ECAI-2012, so that OC roles will be well-prepared, based on the 2007-2011 experience. Corporate know-how of the RuleML directors will be available to the thesis developer. The human RuleML-2012 chairs can then fully benefit from the prepared PA knowledge bases, building on and modifying/extending the encoded knowledge. This will also help the OC chairs to select and possibly change roles if they fit into another from the Personal Agent's FOAF-like profile. This answer once received by the OA is then forwarded to the External Agent (EA) for humans users.

The Personal agents used by the SymposiumPlanner contain FOAF-extending profiles for each person of the organizational team. Each PA contains a knowledge base that represents its chair's responsibilities to answer corresponding queries. In these profiles besides FOAF-like facts, person-centric rules are used as well. All facts and rules (clauses) are serialized in Naf Hornlog RuleML [13], the RuleML sublanguage for Horn logic (allowing complex terms) enriched by Naf (Negation as failure). The FOAF extending profiles have access to RDF and RDFS/OWL (role and responsibility models).

SymposiumPlanner has a distributed architecture: instead of saving facts and rules for different PAs at a centralized location. Our system has distributed knowledge bases for the members of an organization. Since, each member is represented by a PA. The PAs have individual knowledge bases containing information regarding the responsibilities of persons being represented by the PA. This architecture enables us to enrich and constantly update knowledge bases for PAs according to our requirements without causing any issues for the other PAs. The information in PA knowledge bases can be improved and revised.

role in a more efficient manner. This gradual adoption will serve to evaluate and tune the system before it is utilized for participant query answering. Our Web interface should allow participants to issue queries via web forms, which will generate both RuleML/XML and controlled natural language. By reusing and integrating existing factual information about hosting ECAI-2012 conference on the Internet, we envision avoiding redundancy in the knowledge bases of the SymposiumPlanner agents 15.

Since there will be only one symposium site in 2012, we will need to reify the 2-SubOrganizationalAgents' business logic of SymposiumPlanner 2011 into one OrganizationalAgent. In order to evaluate the SymposiumPlanner-2012 system, we will solicit feedback from the emerging RuleML-2012 OC, which will lead to continuous revisions of the knowledge bases, and perhaps the SymposiumPlanner-2012 system architecture as an instantiation of Rule Responder. This will then allow further improved designs and implementations for the SymposiumPlanner system of 2013 and beyond. One of the main advantages of SymposiumPlanner is that it answers user's queries and reduces user's burden of finding the relevant information by themselves. The queries include the information about the Symposia. For its usability, the SymposiumPlanner user
5.2 Task Responsibility Matrix

The SymposiumPlanner uses a Responsibility Assignment Matrix (RAM) to support the OA in its selection of a PA and the participating profiles [10]. A RAM describes the responsibility of agent roles in completing certain tasks or deliverables in a virtual organization. A standard RAM is a matrix with [34],

- Responsible: Agents who do the work to achieve a task.
As one possible way for coordination in a virtual organization the Rule Responder framework uses a 'pluggable' Responsibility Assignment Matrix (RAM) to support the OA in its selection of a PA and its optional participating profiles underneath. A RAM describes the responsibility of agent roles in completing certain tasks or deliverables in a virtual organization. A standard RAM is a matrix which describes four key responsibilities most typically used: Responsible, Accountable, Consulted, and Informed.

- Accountable: Agent is accountable for completion of a task.

- Informed: Agent who is kept up-to-date on progress, often only on completion of the task or deliverable.

Since, SymposiumPlanner follows a star-like agent topology, a single matrix can be used in the OA to map an incoming query to the PA whose local knowledge base is deemed to be best suited for answering a query. The matrix is represented as an OWL ontology (OWL Lite) and can be used by the SymposiumPlanner agents via querying using the Semantic Web built-ins of Prova.

In SymposiumPlanner OWL ontology, we have defined certain specialized tasks to their respective agents. This mapping of tasks with agents helps the Mule ESB to assign the incoming query to the right endpoint for processing. We shall now look at the structure of our ontology.

The main class of the ontology is the SymposiumPlannerSystem. All the sub-classes branch out from this main class.

<owl:Class rdf:ID="SymposiumPlannerSystem"/>

The first major sub-class of the SymposiumPlannerSystem class is the FunctionManagement class which is at the root of tasks assigned to different PAs in the SymposiumPlanner.

<owl:Class rdf:ID="FunctionManagement">
<owl:Class rdf:ID="FunctionManagement">
<rdfs:subClassOf rdf:resource=""/>
"#SymposiumPlannerSystem" />
</owl:Class>

The task assignment is typically done through defining a new sub-class of this FunctionManagement class. For example, getContact is used by the SymposiumPlanner to retrieve contact information of a particular agent. This "getContact" class is first declared as a sub-class of the FunctionManagement.

<owl:Class rdf:ID="getContact">
<rdfs:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

As discussed earlier, “responsible” PA agents are mapped to their respective roles and tasks inside the RAM. In our case, since the organizing committee of the RuleML-2012 is to be contacted. This responsibility is mapped in the following way.

<owl:ObjectProperty rdf:ID="responsible">
<rdf:type rdf:resource="owl#FunctionalProperty" />
<rdfs:domain rdf:resource="#FunctionManagement" />
<rdfs:range rdf:resource="&ruleml2012ATecai;RuleML-2012-ECAI"/>
</owl:ObjectProperty>
The “getContact” task is then mapped to the responsible OA ”RuleML-2012-ECAI” which will be able to delegate this task to the PA in we are looking for.

<Get_Contact rdf:ID="getContact">
<responsible rdf:resource="&ruleml2012ATecai;RuleML-2012-ECAI" />
</Get_Contact>

The rest of the tasks are mapped to their respective roles and agents in a similar fashion. The detailed RAM is attached in the appendix.

Sub-OA Responsibility Assignment Matrix

SymposiumPlanner 2012 has two Sub-OAs. While one Sub-OA maps a conference at RuleML, the other Sub-OA maps RuleML as an organization. The Sub-OAs have their own RAMs to help the Sub-OAs delegate tasks to the right PAs once the Super-OA delegates a query on merit to one of the SUB-OAs based on delegation logic in the form of the RAM as well as keywords which help the Super-OA delegate tasks to SUB-OAs.

Lets discuss the structure of these RAMs for the Sub-OAs to understand the structure of organizations being mapped by the Sub-OAs. We will start with the Sub-OA mapping the RuleML conference. The ontology defining the responsibilities of the organizing committee of a conference has a main class called “Organizing Committee”.

<owl:Class rdf:ID="Organizing_Committee"
One of the members of an organizing committee is the publicity chair. This role is declared as a sub-class of organizing committee.

The responsibility domains are defined as follows:

Each member of the organizing committee has certain responsibilities depending on their profile in the organization. These responsibilities are mapped to their respective roles as follows. Firstly, some responsibilities are defined which represent actions that are to be mapped to roles in the organization.
The responsibilities mentioned above are to be mapped to their respective roles which are defined in the ontology as meta topics.

<!-- Meta Topics -->

<Responsibility rdf:ID="DocSymposiumChair" />
<Responsibility rdf:ID="IntlRuleChalSteerCommitChair" />
<Responsibility rdf:ID="IntlRuleChalChair" />
<Responsibility rdf:ID="LocalChair" />
<Responsibility rdf:ID="MetaAndSocialChair" />

The responsibilities are then mapped to their roles (meta topics) in the following way:

<Publicity_Chair
rdf:ID="PublicityChair">
<responsible rdf:resource="#Partners" />
<responsible rdf:resource="#Sponsoring" />
<responsible rdf:resource="#PublicityChair" />
</Publicity_Chair>

<DocSymposium_Chair
rdf:ID="DocSymposiumChair">
<supportive rdf:resource="#Submission" />
<responsible rdf:resource="#DocSymposiumChair" />
</DocSymposium_Chair>
Similarly, in the case of mapping an organization (in our case RuleML) the responsibilities and roles are defined in an owl ontology which is attached to this thesis in the appendices.

5.3 Global Knowledge Base for Virtual Organization

The ontologies and rules are globally shared via the Organizational Agent (OA) to assist all the PAs. Another subset of rules is distributed amongst the PAs as locally distributed knowledge bases. This shared ontology and the shared subset of rules is referred to as the global knowledge base.

Global knowledge in SymposiumPlanner is modeled as a combination of ontologies and rules, where rule arguments are defined by signatures. The rulebase includes a light-weight ontology realizing the Responsibility Assignment Matrix (RAM) which makes it possible for the Sub-OA to delegate tasks to PAs based on the content of the incoming query [15].
The global rules include general constraints and preferences of the virtual organization that is mapped by the SymposiumPlanner. The SymposiumPlanner makes use of the rule format RuleML/XML and the Rule Responder framework to transform to and from other rule languages.

In the SymposiumPlanner, we attempt to define the rules in the global knowledge base using Prova which is a highly expressive Semantic Web rule engine. Prova provides support for two external type systems, namely Java class hierarchies and ontological type systems (e.g. OWL or RDFS ontologies) respectively Description Logic knowledge bases. Prova provides a rich library of built-ins for query languages such as SQL, SPARQL, and XQuery [9].

The following rule uses a SPARQL query built-in to access an RDF Friend-of-a-Friend(FOAF) profile published on the Web. The selected data is then assigned to variables which can be used within an agent’s rule logic, e.g. to expose the accepted trackpapers.

```
getTrackPaper("accepted", Paper) :-
  QueryString = 'PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
SELECT ?paper ?title
FROM <http://localhost:8080/>
```
Where {
?paper a ?type .
FILTER (?type = <http://ruleml.org/ontology#TrackPaper> ) .
}

Each Prova knowledge base file is pointed to by the Mule config file (mule-config.xml).

**OWL Ontology Assignment**

A fact is used to point Prova to the OWL ontology used by the particular Sub-OA. The OWL ontology represents a responsibility assignment matrix which is used by the OA to map an incoming query to the PA whose local knowledge base is deemed to be most suitable for answering it. The assignment of the ontology in the global knowledgebase is implemented as follows, e.g. import("http://localhost:8080/RuleML-2012-ECAI/ECAI2012.owl").

**Interfaces**

Interfaces describe what queries are expected to be issued to the Symposium-Planner. They have the following format:

interface(performative(Performative), performative("?"),"description").
To further elucidate the arrangement, let us look at an interface for contact information.

interface(getContact(Topic, Request, Contact),
getContact("+","+","-"), "request personal contact information for a certain topic and request regarding RuleML-2012").

- The first argument is the performative.

interface(getContact(Topic, Request, Contact))

- The second argument is the relation with +’s and -’s to represent constants and variables.

getContact("+","+","-"")

- The last argument is the description of the framework.

"request personal contact information for a certain topic and request regarding RuleML-2012"

processMessage Rule

An incoming query is handled by the processMessage rule. The structure of the processMessage rule is as follows:

processMessage(XID, From, UserName, performative(Performative)):-
- XID - The name of the OA.

- From - The name of the endpoint.

- Username - Primitive (e.g. User).

- Performative - The relation name surrounded by Rel.

- Performative - The contents of the relation (Ind’s, Vals’s, Expr’s, Plex’s, etc.)

A rule is then added to the ”look up” the responsible Agent:

\[\text{assigned(XID,Agent,Responsibility,Role)}\],

where,

- XID - The name of the OA.

- Agent - Name of the found agent.

- Responsibility - A responsibility name for the query.

- Role - What kind of role the responsibility will be matched to.

For communication between distributed agents Prova supports special built-ins for asynchronously sending and receiving messages within serial Horn rules. The main constructs of messaging reaction rules are:

- sendMsg predicate: To send messages.
- rcvMsg: Reaction rules which react to inbound messages.

- rcvMult: inline reactions in the body of messaging reaction rules to receive one or more multiple inbound event messages.

A query is then sent to the agent that was found:

sendMsg(XID,esb,Agent,Type,performative(Performative)),

Here,

- XID - The name of the OA.

- esb - Transport protocol.

- Agent - Name of the found Agent.

- Type - query

- performative - The relation name of the query.

- Performative - The contents of the relation.

The answer is then received from the Agent:

rcvMult(XID,esb,Agent,Type,performative(Performative)),

Here,

- XID - The name of the OA.

- esb - Transport protocol.

- Agent - Name of the found Agent.
- Type - answer.

- performative - The relation name of the query.

- Performative - The contents of the relation.

This answer is then sent back to the External Agent (EA):

```plaintext
Here, sendMsg(XID,esb,From,"answer", performative(Performative)),
```

- XID - The name of the OA.

- esb - Transport protocol.

- Agent - Name of the EA endpoint.

- Type - answer.

- performative - The relation name of the query.

- Performative - The contents of the relation.

Note: Performative describes the pragmatic envelope for the message content. A standard nomenclature of performatives is, e.g., the FIPA Agents Communication Language (ACL) [6].

5.4 FOAF

To a computer, the Web is a flat, boring world, devoid of meaning. This is a pity, as in fact documents on the Web describe real objects and imaginary
concepts, and give particular relationships between them. For example, a document might describe a person. The title document to a house describes a house and also the ownership relation with a person. Adding semantics to the Web involves two things: allowing documents which have information in machine-readable forms, and allowing links to be created with relationship values. Only when we have this extra level of semantics will we be able to use computer power to help us exploit the information to a greater extent than our own reading. - Tim Berners-Lee “W3 future directions” keynote, 1st World Wide Web Conference Geneva, May 1994 [5].

FOAF describes the world using simple ideas inspired by the Web. In FOAF descriptions, there are only various kinds of things and links, which we call properties. The types of the things we talk about in FOAF are called classes. FOAF is therefore defined as a dictionary of terms, each of which is either a class or a property. Other projects alongside FOAF provide other sets of classes and properties, many of which are linked with those defined in FOAF. FOAF descriptions are themselves published as linked documents in the Web (eg. using RDF/XML or RDFa syntax). The result of the FOAF project is a network of documents describing a network of people (and other stuff). Each FOAF document is itself an encoding of a descriptive network structure. Although these documents do not always agree or tell the truth, they have the useful characteristic that they can be easily merged, allowing partial and decentralised descriptions to be combined in interesting ways [33].

FOAF contains a massive list of classes and properties, however we shall list...
the ones we have used in our system.

- foaf:homepage “homepage - A homepage for some thing. ”
- foaf:made “Something that was made by this agent.”
- foaf:mbox “A personal mailbox, ie. an Internet mailbox associated with exactly one owner, the first owner of this mailbox.”
- foaf:member
- foaf:based near “based near - A location that something is based near, for some broadly human notion of near. ”
- foaf:firstName “firstName - The first name of a person. ”
- foaf:mboxsha1sum “sha1sum of a personal mailbox URI name - The sha1sum of the URI of an Internet mailbox associated with exactly one owner, the first owner of the mailbox.”
- foaf:name “name - A name for some thing. ”
- foaf:phone “A phone, specified using fully qualified tel”
- foaf:title “title - Title (Mr, Mrs, Ms, Dr. etc) ”
5.5 Locally Distributed Knowledge Bases for Personal Agents

The Friend of a Friend (FOAF) project is creating a Web of machine-readable pages describing people, the links between them and the things they create and do. (www.foaf-project.org/about).

In SymposiumPlanner we utilize a single organizational agent to handle the filtering and delegation of incoming queries. Each personal agent (PA) acts in a rule-governed manner on behalf of the person or entity being represented by the PA. Each agent manages personal information, such as FOAF-like profile containing a layer of facts about the person/entity being represented by the profile. These rules allow the PA to automatically respond to requests concerning the Symposium.

Let’s see some of the FOAF-like profiles of Personal Agents to establish a better understanding of our methodology. We start with a basic profile like that of the PA representing the General Chair of a Symposium Organizing Committee.

```prolog
person(
    symposiumChair[ECAI_2012, general],
    foafname[firstName[Guido], lastName[Governatori]],
    foaf_title[title[Dr]],
    foaf_email[[]],
    exphones[telephoneNumbers[office[], cellPhone[]]]).
```
- We are describing a person who is a "General Chair" of the Symposium at ECAI.

- The first name and last name are packaged into the category foafname.

- The title of the person in question is part of the foaftitle.

- The email of the person is recorded inside the foafmbox.

- The phone extensions can be recorded inside exphones.

In order to gain a better understanding of rule and facts in our FOAFextended profiles (local knowledge base) for personal agents. We shall dis-cussthe profile of the Publicity Chair. In the beginning we establish that the Publicity Chair of the Symposium at ECAI is Dr. Frank Olken.

person(
symposiumChair[RuleML_2012,publicity],
foafname[firstName[Frank],lastName[Olken]],
foaftitle[title[Dr]],
foafmbox[email[folkenATnsfDOTgov]],
exphones[telephoneNumbers[office[7032927350],cellPhone[]]])

- Line 1 describes the role of the person in question as in Publicity Chair at RuleML-2012.

- FOAFName: First Name - Frank, Last Name - Olken.
The profile also contains a sponsoring rule for the RuleML-2012 which we shall discuss now. The sponsoring mechanism allows the conference to be able to give dividends based on sponsoring levels of a contributing organization/person.

\[
sponsor(\text{contact}[\text{?Name,?Organization}], \\
\text{?Amount:integer},\text{results}[\text{?Level,?Benefits,?DeadlineResults}]
, \text{performative[?Action]}) :- \\
\text{requestSponsoringLevel(\text{?Amount:integer,?Level})}, \\
\text{requestBenefits(\text{?Level,?Benefits})}, \\
\text{checkDeadline(\text{?DeadlineResults})}, \\
\text{checkAction(\text{?Action,?Level,?Amount:integer})}.
\]

The query requires that:

\[
\text{?Amount:integer}
\]

- Amount the organization is donating.

\[
\text{results[?Level,?Benefits,?DeadlineResults]}
\]
- The results contain the level of the donations.
- The benefits of the donations.
- If the deadline has passed or not.

performative[?Action]

- The performative contains an action that should occur from the result of the donation.

The result of the query depends on the outputs froms requestSponsoringLevel, requestBenefits, checkDeadline and checkAction. So we shall now discuss how those parts of the rule individually work on producing an output.

checkAction(?Action,?Level,?Amount:integer) :-
actionPerformed(?Action,?Level,?Amount:integer).

Check action will determine what action the Publicity Chair should do depending on the level of the donation. This outcome depends on the result of actionPerformed. The actionPerformed rule has further rules to cover sponsorship amount based rules and action.

- When a sponsor makes a donation under 500 the Publicity Chair encourage them to donate more.

actionPerformed(?Action:string,preSponsor,
?Amount:integer):- subtract(?Result:integer,
500:integer,?Amount:integer),
stringConcat(?Action,?Result:integer).
presponsor(encourage[donate[300]]).

- When a sponsor makes a bronze donation the Publicity Chair should email the organization.

  actionPerformed(email,bronze,?Amount:integer).

- When a sponsor makes a silver donation the Publicity Chair should email the organization.

  actionPerformed(email,silver,?Amount:integer).

- When a sponsor makes a gold donation the Publicity Chair should phone the organization

  actionPerformed(phone,gold,?Amount:integer).

- When a sponsor makes a platinum donation the Publicity Chair should phone the organization

  actionPerformed(phone,platinum,?Amount:integer).

- When a sponsor makes a emerald donation the Publicity Chair should phone the organization

  actionPerformed(phone,emerald,?Amount:integer).
The publicity chair rule-base also checks whether the deadline has passed or not by comparing the current date to the deadline date. If the date is greater than the deadline date, then the deadline has past.

\[
\text{checkDeadline(passed[deadline]):-}
\]
\[
\text{date(?X:integer),}
\]
\[
\text{deadline(sponsoring, ?D:integer),}
\]
\[
\text{greaterThan(?X:integer,?D:integer).}
\]

This rule checks to see if the deadline is on going. It must compare the current date to the deadline date. If the date is less than the deadline date, then the deadline is on going.

\[
\text{checkDeadline(onGoing[deadline]):-}
\]
\[
\text{date(?X:integer),}
\]
\[
\text{deadline(sponsoring, ?D:integer),}
\]
\[
\text{lessThan(?X:integer,?D:integer).}
\]

The publicity chair rule-base also contains facts some of which shall be discussed here. For example, we are defining the benefits a sponsor receives based on the sponsoring level through the following facts.

- A Pre-Sponsor level, means there will be no benefits to offer.

\[
\text{benefits(preSponsor, benefits[none]).}
\]
- A bronze level of sponsoring means that we shall allow the sponsor to put a logo on our site and also acknowledge the sponsor during the course of proceedings.

```scheme
benefits(bronze, benefits[ logo[on[site]], acknowledgement[in[proceedings]] ]).```

- A silver sponsor get the following benefits:
  1) Logo on Site
  2) Acknowledgement in Proceedings
  3) Option to target sponsoring Student

```scheme
benefits(silver, benefits[ logo[on[site]], acknowledgement[in[proceedings]], option[sponsor[student]] ]).```

- A gold sponsor get the following benefits:
  1) Logo on Site
  2) Acknowledgement in Proceedings
  3) Option to target sponsoring Student
  4) 1 Free Registration
  5) Logo in Proceedings
  6) Option to give out demos
We shall now discuss another form of rule that is also part of the Publicity Chair profile. This rule deals with partner organizations who have partnered with the conference in terms of donations or publicity. The meetingPartner relation displays the meeting and the name of the partner.

\[
\text{meetingPartner(?Meeting, ?PartnerName) :-}
\]

The rule above depends on facts detailing the partner organization's meeting and URI. The organization's name is shown based on URI and name. For example, one of the partner organizations of RuleML 2012 is Oasis. The facts are shown in the following way,

\[
\text{partnerOrganization(RuleML_2012, "http://www.oasis-open.org/").}
\]
\[
\text{organizationName("http://www.oasis-open.org/", oasis).}
\]
As a result the meetingPartner relation will output the name of the meeting as well as the name of the partner.

Similarly, we have another rule regarding the sponsors. This rule shows us a sponsoring organization based on the information regarding their sponsoring levels. As a result, the output contains information regarding the meeting, sponsor and their sponsoring level.

\[
\]

The above rule is supported by facts. For example University of Bologna has a gold sponsoring level. The facts also detail the URI of the University of Bologna which is output as a result of the relation of viewing sponsors.

\[
\text{sponsor(RuleML_2012, uni_bologna, gold).}
\]
\[
\text{sponsorURI(uni_bologna, "http://www.cirsfid.unibo.it/").}
\]

We have another rule for media partners based on information regarding their services, contact and knowledge disciplines that best match their commercial profiles.

\[
\]
The above rule returns information regarding the media partner’s name, media partner's service, media partner’s contact and media partner’s knowledge discipline. The rule is supported by the following facts for example purposes,

mediaPartSectMeeting(RuleML_2012, sigart).
mediaPartnerContact(sigart, "frawley@acm.org").
mediaPartnerDiscipline(sigart, "Artificial Intelligence").

The facts above detail that media partner Sigart is a media partner at RuleML-2012, the contact detail of sigart is an email contact ”frawley@acm.org”. The corporate level discipline of sigart is Artificial Intelligence as they seem to offer a lot of services based on the discipline from seminars to books etcetera.

5.5.1 Processing of Facts and Rules in a Personal Agent Profile

In the previous section, we saw how the Personal Agent profiles contain knowledge about a personal agent. In this section, we will discuss the processing of those rules and facts in the Prova knowledge base. The Sub-O rganizational Agents are implemented in Prova. We shall now discuss the
structure of a Sub-Organizational Agent prova knowledge base to understand the functionality of the Sub-OAs.

Prova supports rulebase import. The imported rulebases are managed as modules in the knowledge base. Multiple nested imports are possible. A typical import is made possible in the following way,

```prova
:-eval(consult('../../ContractLog/math.prova')).
:-eval(consult('../../ContractLog/datetime.prova')).
:-eval(consult('../../ContractLog/list.prova')).
:-eval(consult('../../ContractLog/update.prova')).
:-eval(consult('../../ContractLog/utils.prova')).
```

In above snippet, the prova KB is importing multiple prova rulebases for use during decisioning at the knowledge base level.

The Prova KB also describes the pragmatic envelope for the message content using performatives. In case there is a request from the EA, the OA will issue a response.

```prova
performative(request):-performative(query).
performative(query).
```

```prova
performative(XID,Performative):-
    performative(Performative).
```

In SymposiumPlanner each PA runs a rule engine which accesses different sources of local data and computes answers according to the local rule-based
decision logic of the PA. Arbitrary rule engines can be used by the PA as long as they have an interface to ask queries and receive answers which are translated into common Reaction RuleML interchange format in order to communicate with other agents. Reaction RuleML provides an interface definition language which allows descriptions of the signatures of publicly accessible rule functions as well as their modes and type declarations. Modes are states of instantiation of the predicate described by mode declarations.

. ”+” The term is intended to be input.

. ”-” The term is intended to be output.

. ”?” The term is undefined/arbitrary (input or output).

External agents can access the system only via public interfaces, which reveal abstracted information to authorized users and hide local information of the organization. An interface defined in the Sub-Organizational Agent for the conference at RuleML 2012 is,

```interface(getContact(Topic,Contact),
getContact("+","-"),"request personal contact information for a certain Topic and Request regarding RuleML-2012@ECAI")```

which is able to handle a request like,
This interface is designed as a response to the query which is issued from the External agent. The interface is able to match the relation with the topic and the request for retrieving contact information for the topic in question. This topic once mapped to the responsibility assignment matrix is able to send it to the Sub-OA which contains the interface to handle this request. In the request above, the topic(ruleml2012ATecaiGeneralChair) is intended for input "+" so that it can be mapped through the responsibility assignment matrix to the appropriate Sub-OA. The variable "Contact" is meant to be intended for output "-".

5.5.2 Illustrative Example

In the previous section, we discussed a rule in the PA local knowledge base regarding media partners.

meetingPartner(?Meeting, ?PartnerName) :-
partnerOrganization(?Meeting, ?PartnerURI),
organizationName(?PartnerURI, ?PartnerName).
The ProcessMessage(...) rule is used to process the user’s query. It has the following parameters: (a) XID is the name of the OA (e.g., TeamFormer), (b) From is the name of the endpoint, (c) Primitive is the user name (e.g., User), (d) X is the relation name surrounded by angle brackets as in <Rel>, (e) Args is the arguments of the relation (<Ind>, <Var>, <Expr>, <Plex>, etc.). When a processMessage(...) is fired, its body is checked; if the rule body is true, the processMessage(...) ends with fail() to prevent any further processing of the query.

```
<RuleML xmlns="http://www.ruleml.org/0.91/xsd"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.ruleml.org/0.91/xsd
                              http://ibis.in.tum.de/research/ReactionRuleML/0.2/rr.xsd"
         xmlns:ruleml2012="http://ibis.in.tum.de/projects/paw#">
  <Message mode="cutbound"
            directive="query-sync">
    <oid>
      <Ind>SymposiumPlannerSystem</Ind>
    </oid>
    <protocol>
      <Ind>esb</Ind>
    </protocol>
    <sender>
      <Ind>User</Ind>
    </sender>
    <content>
      <Atom>
        <Rel>mediapartnerWithSector</Rel>
        <Var>Meeting</Var>
        <Var>Partner</Var>
        <Var>Service</Var>
        <Var>Contact</Var>
        <Var>CategoryofKnowledges</Var>
      </Atom>
    </content>
  </Message>
</RuleML>
```

Figure 5.2: Media Partner Query
Figure 5.3: Media Partner Query Response

In the Sub-Organizational agent of SymposiumPlanner, the above message is handled by a prova rule which is as follows,

```prolog
processMessage(XID, From, Primitive, meetingMediaPartner(Meeting, PartnerName)) :-
    !,
    println(["---------------------------------"],
    println([" QUERY RECEIVED "]],
    println(["---------------------------------"],
    println(["Received message from browser."]],
    println(["Looking up Responsible Personal Agent"]],
    The processMessage rule is used to process the user’s query.
```

79
Let us now look at the processMessage function body,

- XID : Conversation ID.
- From : Who is sending this message (can be an agent or a service).
- Primitive: Pragmatic instruction, FIPA ACL primitive. (Pragmatic characterization of the message context broadly characterizing the meaning of the message).
In the assigned part the rule looks at the conversation id, the agent who is capable of answering this question i.e the agent who is responsible for maintaining information on partners. In response the sendMsg() asks the agent who was discovered above to answer the question. If the answer is more than one then rcvMult() allows the personal agent to respond with multiple responses to the above question. Since, there are more than one partner, the substitutions allow for multiple results to be displayed in the output.

5.6 Mule ESB

Mule ESB is a lightweight Java-based enterprise service bus (ESB) and integration platform that allows developers to connect applications together quickly and easily, enabling them to exchange data. Mule ESB enables easy integration of existing systems, regardless of the different technologies that the applications use, including JMS, Web Services, JDBC, HTTP, and more. The key advantage of an ESB is that it allows different applications to communicate with each other by acting as a transit system for carrying data between applications within your enterprise or across the Internet.

Mule ESB includes powerful capabilities that include:
- Service creation and hosting expose and host reusable services, using Mule ESB as a lightweight service container

- Service mediation shield services from message formats and protocols, separate business logic from messaging, and enable location-independent service calls

- Message routing route, filter, aggregate, and re-sequence messages based on content and rules

- Data transformation exchange data across varying formats and transport protocols. (http://www.mulesoft.org).

Mule provides a distributable object broker to manage all sorts of service
components such as the agent services in our system. The three processing modes of Mule ESB are,

- Asynchronous: Many messages can be processed by the same component at a time in various threads.

- Synchronous: The whole request is processed in a single thread.

- Request-Response: The component is able to make a specific request or an event and wait for a specified time to get a response back.

The object broker follows the Staged Event Driven Architecture (SEDA) pattern [32]. The basic approach of SEDA is to decompose a complex, event-driven application into a set of staged connected by queues. SEDA supports massive concurrency demands on Web-based services and provides a highly scalable approach for asynchronous communication. Distributed Agents which are running a rule engine are deployed as Mule components which listen at configured endpoints e.g. JMS endpoints. Since, Reaction RuleML is the system-wide rule interchange language between the agents. Translator services are used to translate inbound and outbound messages from Reaction RuleML into platform specific execution syntaxes of rule engines. In our case XSLT based translator services are provided as Web forms.

The main parts of Mule architecture are,

- Component: This part contains the business logic e.g. a bean, a service or a POJO (Plain Old Java Object).
- Transport: This part handles connectivity with a specific technology e.g. JMS, SAP etc.

- Transformer: This part transforms the data to the format the next part can understand.

- Inbound Routers: This part determines what to do with the received message before it is sent to the service.

- Outbound Routers: This part determines where a message needs to be sent to after it has been processed by the service.

The steps involved in this whole process are as follows,

- A transport receives a message.

- The message is transformed to the required format and forwarded to the inbound router.

- The message is processed by the inbound router.

- The inbound router forwards this message to the component which applies its business logic.
- After application of business logic, the message is forwarded to the outbound router.

- The message is then forwarded to the target (configured in the configuration file).

To further elucidate how endpoints are configured in our system. Let us discuss the mule-config (Mule Configuration file). We have configure the environment properties detailing the address of our tomcat installation as well as the port assigned to the Mule.

```xml
<global-property name="http.host" value="localhost"/>
<global-property name="MULE_PORT" value="8888" />
<global-property name="MULE_IMPLEMENTATION" value="ws.prova.mule.impl.ProvaUMOImpl"/>
<global-property name="TOMCAT" value="http://127.0.0.1:8080" />
<global-property name="jms.url" value="vm://localhost"/>
```

In order to create endpoints for Personal Agents, we must characterize them as topics. Each Personal Agent is response for a topic regarding which the PA is best suited to answer a query. In the case of RuleML-2012, we implemented the topic names as,

```xml
<!-- ECAI-2012 -->
<global-property name="ECAI2012EAName" value="ECAI-2012" />
<global-property name="ECAI2012OAName" value="ECAI2012" />
```
In our system the main transport protocol is JMS which is configured here.

```xml
<jms:activemq-connector name="jmsConnector"
    specification="1.1" brokerURL="${jms.url}" />
```

The Mule configuration also contains service endpoints.

```xml
<!-- service endpoints of the ECAI-2012 use case -->
```
The endpoint descriptors are used to create a model for the Mule configuration to allow communication between External Agents and that specific
model for its underlying topics (i.e. Personal Agents).

```xml
<model name="ECAI-2012Model">
  <service name="ECAI-2012Service" initialState="started">
    <inbound>
      <jms:inbound-endpoint ref="RuleML-2012-ECAI">
        <properties>
          <!-- Each receiver polls with a 5 second interval -->
          <spring:entry key="rulebase" value="${ECAI2012_Prova}" />
        </properties>
      </jms:inbound-endpoint>
      <http:inbound-endpoint
        address="http://${http.host}:${ECAI2012_PORT}" />
    </inbound>
    <component class="${MULE_IMPLEMENTATION}" />
  </service>
</model>
```

The large variety of transport protocols provided by MULE can be used to transport the messages to the registered endpoints. In our case JMS is used for internal communication, while HTTP and SOAP are used for external Web access. The usual processing style of MULE is based on SEDA event queues but MULE can sometimes use the synchronous communication to handle communication with HTTP clients such Web browsers. In this case, a synchronous bridge component dispatches the requests into the
asynchronous messaging framework and collects all answers from the internal service nodes. All the answers have been collected, they are sent back to the still connected external service via the HTTP-synchronous channel.
Chapter 6

Case Study:

SymposiumPlanner-2012 Query Based Conference Management System

In this chapter we take the latest SymposiumPlanner 2012 for example. SymposiumPlanner 2012 inherits the basic functions of the previous SymposiumPlanner instantiations, but improves on them. Its support builds on earlier SymposiumPlanners and an extensive literature survey, distilling design principles and implementation techniques for future SymposiumPlanners from the lessons learned so far. SymposiumPlanner 2012 provides a novel architecture, newer complex queries and a user friendly user-client.
The SymposiumPlanner 2012 unlike past is covering only one instalment i.e the 6th International Symposium on Rules, RuleML 2012 which is being held at ECAI (Montpellier, France). The SymposiumPlanner uses two OAs while one deals with the Symposium at ECAI, the other maps the RuleML organization. For the first time the SymposiumPlanner not only maps a conference but also maps an organization into the Rule Responder architecture.

A super OA delivers and filters queries and requested tasks to the sub OAs, which act as a single point of entry for each part of the SymposiumPlanner. Sub-OAs keep on filtering, deciding and delegating incoming queries to the organization’s members which are implemented as distributed rule-based personal agents.

Sub-OAs keep on filtering, deciding and delegating incoming queries to the organization’s members which are implemented as distributed rule-based PAs. Responsibility assignment Matrix (RAM) is used to describe the roles and responsibilities of the sub-OAs and PAs in the virtual organization of RuleML Symposium 2012. Negotiation and distributed coordination protocols are applied to manage and communicate with the project team and external agents. The PAs act as self-autonomous agents having their own rule-based decision and behavioural logic on top of their personal information source, Web services, vocabularies/ontologies and knowledge structures. In SymposiumPlanner 2012, we use rule agent Prova for implementing the OAs. Prova packs lot of a features, some of them inspired from latest developments in modern computation logic, functional programming, distributed systems, and event
driven architectures. The rule agents consult knowledge from repository as well as sparql to access external data to reduce redundancy.

Meanwhile, we use Mule ESB 3.1, which includes improvements in Cloud Connect, including custom schemas for each connector, much simpler invocation of connectors, a new polling mechanism, message enrichment capabilities, and a simple yet powerful logging facility. Reaction RuleML messages are transported via the ESB to the appropriate agents or external communication interfaces based on a broad spectrum of transport protocols which can be selected based on their merits and usage. For a better visualization of the architecture, see 6.1.
6.1 Distributed Rule System vs. a Centralized Rule System

Our system is implemented as a distributed rule system. It connects OAs and PAs so that they can share their knowledge. The External Agent can query knowledge both in OAs and PAs. The PA’s rules (a FOAF-extending profile) correspond to their expert owner while the OA’s knowledge describes the virtual organization as a whole. The PAs along with their rule bases are stored at distributed locations.

In contrast, a centralized system would contain all the knowledge (facts and rules) in a centralized location. So, all of the knowledge would be contained in a single file or database. The advantages of a distributed rule system over a centralized system include ease of maintenance, achieving a fault-tolerant system by using distribution for redundancy, and improved efficiency through distributed processing.

Distributed maintenance allows agents to update their rules and facts without affecting the rule bases of other agents. If all of the knowledge was stored in one central rule base, problems introduced by one agent would affect the entire system. Distributed maintenance has proved useful during the development of SymposiumPlanner 2012 usecase. If an agent is not running, it does not affect the system performance as all agents do not need to be working for the system to perform some action. If the PA to whom a question is addressed is not working, the OA will try to contact the PA. If the OA
Figure 6.1: SymposiumPlanner 2012
does not receive any response, it will request a timeout and try to look for other PA(s) likely to answer a question. If the OA is not able to find anyone to answer the query, the system will tell the user that the PA (needed to answer this query) is offline.

The distribution of rule-bases could affect the overall performance of the system as the system has to go for frequent timeouts to look for the ideal PA able to answer a question but the overhead is not noticeable to external agents. Rules execute faster when there are less clauses for the engine to process. Our approach gives a robust solution which ensures the system is more fault tolerant. By introducing distributed rule bases, we are able to ensure the system is less prone to problems and with more rule engines translating knowledge bases instead of one rule engine reading a massive knowledge base the processing is faster.

6.2 Organizational Agent

An Organizational Agent is used to describe the goals of the organization as a whole and contains a knowledge base that describes the symposium’s policies, regulations, and opportunities. This knowledge base contains condition/action/event rules as well as derivation rules. An OA manages its local PAs, providing control of their life cycles and ensuring overall goals and policies of the organization and its semiotic structures [10]. An example query that the OA can answer for the RuleML-2012 Sympo-
sium is "Who are the media partners of the RuleML-2012 and what are the services that their offer along with their contact details and can you also give me information of knowledge expertise of partners?". When a RuleML-formalized version of this query is received by the OA, this agent must first determine which PA is capable of answering this query.

The OA cleverly also uses decisioning mechanism of which Sub-OA to channel the question to. The Role Assignment Matrix can help the OA in determining which PA under a specific Sub-OA is capable of answering the query. The Sub-OA then determines which PA contains the necessary information to answer the query. When the correct PA is determined, the OA delegates the query to that member’s PA. The PA will then respond with the informa-
tion from their FOAF-like profile. In this case, a listing of media partners along with their services, contact and knowledge expertise.

OAs can act as a single point of entry to the managed sets of local PAs to which requests by EAs are disseminated. This allows for efficient implementation of various mechanisms of making sure the PAs functionalities are not abused (security mechanisms) and making sure privacy of entities, personal data, and computation resources is respected (privacy & information hiding mechanisms) [10]. The selection logic for the dissemination of queries to PAs is described by Responsibility Assignment Matrix (RAM) and OA selects responsible agents. The Responsibility Assignment Matrix is a method that the OA utilizes for query delegation. SymposiumPlanner 2012 has two OAs. While one deals with the 6th International Symposium on Rules (http://dbis.informatik.tu-cottbus.de/ruleml2012/), the other OA maps RuleML Organizational structure (http://ruleml.org/).

Organizational Agents are implemented using the Prova Semantic Web rule engine. Prova messaging reaction rules are used for managing the communication flows in the OAs. A coordination pattern is implemented into the OA as a messaging reaction rule, for an anticipated query from the External Agent (EA). This incoming query is then delegated to the responsible Personal Agent.

**Types of OAs** The SymposiumPlanner has two types of OAs.

- Super Organizational Agent (Super OA) : This OA is at the top of the architectural hierarchy. The Super OA must decide which Sub-OA is
best suited for delegating the incoming query. The Super-OA is able to make a decision regarding a Sub-OA by analyzing the terms used in the query.

processMessage(XID, From, Primitive, [Function|Arguments]):-
  concat([sps_, Function], FunctionwithNamespace),
  assigned(XID, FunctionwithNamespace, "sps_responsible", Agent, Result),
  CandidateSubOAs=ws.prova.mule.impl.DecisionCriteria.subOADecision(Result, Function, Arguments),
  element(CandidateSubOA, CandidateSubOAs),
  sendMessage(XID, esb, CandidateSubOA, "query", [Function|Arguments]),
  % receive answers multiple times
  rcvMult(XID, esb, CandidateSubOA, "answer", Answer),
  sendMessage(XID, esb, From, "answer", Answer).

The decision criteria is based on location and specific terms in the incoming query which leads to successful selection of the desired Sub-OA which can further delegate the query to a PA.

- Sub Organizational Agent (Sub OA): This OA represents a single organization or conference that it maps into a virtual setting. The Sub OA
is aware of the capabilities of the PAs and is able to filter the incoming query to the correct PA.

6.3 Responsibility Assignment Matrix

The SymposiumPlanner uses a Responsibility Assignment Matrix (RAM) to support the OA in its selection of the desired PA. A RAM describes the responsibility of agents in completing tasks in an organization. A standard RAM contains,

. Responsible: Agents who actually will do the work to achieve the set task.

. Accountable: Agents who are accountable for the correct completion of a task.

. Informed: Agents who are kept up-to-date on progress (mostly on completion of a task).

A RAM showing different role descriptions is shown,

Table 6.1: Role Assignment Matrix

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>General Chair</th>
<th>Program Chair</th>
<th>Publicity Chair</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symposium</td>
<td>responsible</td>
<td>consulted</td>
<td>supportive</td>
</tr>
<tr>
<td>Website</td>
<td>accountable</td>
<td>responsible</td>
<td></td>
</tr>
<tr>
<td>Sponsoring</td>
<td>informed, signs</td>
<td>verifies</td>
<td>responsible</td>
</tr>
</tbody>
</table>
The RAM is represented as an OWL ontology (OWL Lite) and can be used by the SymposiumPlanner agent via Prova. The agent’s rule logic binds roles and responsibilities of agents to typed variables in the rule logic. The RAM used by the SuperOA contains information regarding capabilities of Sub-OAs to answer queries.

```xml
<MeetingMediaPartner rdf:ID="meetingMediaPartner">
  <responsible rdf:resource="&ruleml2012ATecai;RuleML-2012-ECAI" />
</MeetingMediaPartner>
```

However, in the Sub-OA the RAM based delegation is based on roles and responsibilities of PAs.

```xml
<Local_Chair rdf:ID="LocalChair">
  <responsible rdf:resource="#RoomManagement" />
  <responsible rdf:resource="#RoomReservation" />
  <responsible rdf:resource="#LocalChair" />
</Local_Chair>
```

### 6.4 Personal Agents

In the SymposiumPlanner system, each organization committee chair is designed as a Personal Agent, which contains a knowledge base that represents its chair’s responsibilities to answer corresponding queries. Personal agents
are chair’s roles in the symposium organization. Personal Agents contain FOAF-extending profiles for each person of the organizational team. Beyond FOAF-like facts, person-centric rules are used. All clauses (facts and rules) are serialized in Naf Hornlog RuleML [18], the RuleML sublanguage for Horn logic (allowing complex terms) enriched by Naf (Negation as failure). These FOAF-extending profiles have access to RDF and RDFS/OWL (role and responsibility models). The SymposiumPlanner 2012 assists each organization by an implemented personal agent. A PA runs a rule engine which accesses different sources of local data and computes answers according to the local rule-based decision logic of the PA. The PAs are secure from the public interface through an intermediate layer of decisioning i.e. the OA layer. The OAs are responsible for filtering queries to the PAs according to their capability to answer a query. Each PA contains a knowledge base that represents its chair’s responsibilities to answer corresponding queries. For example, the query ”What benefits would I receive for sponsoring the symposium with 500 dollars as opposed to 1000 dollars” will be delegated to the Publicity Chair’s agent because it deals with sponsoring of the symposium.
Query Answering for Personal Agents

In some cases, the OA can try to solve a query from an external agent by
itself, but in the following we consider only cases where it delegates queries to PAs. When a PA receives a query, it is responsible for its answering. If there are multiple solutions to a query, the PA attempts to send an enumeration of as many of the solutions to the OA as possible (it is of course impossible when there are infinitely many solutions). There are different methods for processing multiple solutions to a query. A naive method of the PAs would be to first compute all of the solutions and then send all of the answers back to the OA, one at a time. After the last answer message is sent, an end-of-transmission message is sent to let the OA know that there will be no more messages. The main problem with computing all of the answers before sending any of them is obvious: in case of an infinite enumeration of solutions the OA will not receive any answer. The way our implementation addresses the infinite solutions problem is to interleave backtracking with transmission. When a solution is found, the PA immediately sends the answer, and then begins to compute the next solution while the earlier answer is being transferred. When the OA has received enough answers from such a (possibly infinite) enumeration, it can send a no-more message to the PA, stopping its computation of further solutions. Once all solutions have been found in a finite interleaved enumeration, the PA can send an end-of-transmission message.

If a PA receives a query and the agent does not have any solutions for it, a failure message is sent right away back to the OA. If this situation or a timeout occurs (i.e., the PA is offline and did not respond back to the OA
within the preset time period), then the OA can try to delegate the query to
another PA to see if it is able to solve the query. If no solution can be found
in any of these ways, a failure message is sent back to the external agent
that states that the OA (representing the entire organization) cannot solve
the query.
Translation between Agents All the agents are capable of using local languages and engines. However, all rulebases, queries and answers have to be translated into RuleML for transmitting them to other agents. In our case Reaction RuleML provides the translator service framework with Web form interfaces accepting predefined selection based rule templates for the communication with external agents (humans) as well as Servlet HTTP interfaces and Web service SOAP interfaces, which can be used for translation into and from languages such as Prova.

Queries to the SymposiumPlanner are formulated in form of templates which are translated into a discourse representation structure (DRS). The DRS gives a logical/structural representation of the text. It is then fed into an XML parser which translates it into a domain-specific Reaction RuleML representation of the query. Besides parsing and processing the elements of the DRS, the parser also employs transformation rules to correctly translate the query into a public interface that is supported by an OA.

The translator services are using different translation technologies such as XSLT stylesheet, JAXB, etc. to translate from and to Reaction RuleML (which is the general interchange format).

```xml
<Rule style="active|messaging|reasoning"
eval="strong|weak|defeasible|fuzzy">
<oid> <!-- object id --> </oid>
<label> <!meta data of the rule --></label>
<scope> <!-- scope of the rule --> </scope>
```
Reaction RuleML provides a general message syntax for communication between distributed rule-based agents.

Following the above template a real message looks like this,

```xml
<Message>
  <oid> <!-- Conversation id --> </oid>
  <protocol> <!-- Used Protocol --> </protocol>
  <agent> <!-- sender/receiver agent/service --> </agent>
  <directive> <!-- pragmatic primitive --> </directive>
  <content> <!-- Message payload --> </content>
</Message>
```

"http://www.ruleml.org/0.91/xsd"

"http://www.w3.org/2001/XMLSchema-instance"
<Message mode="outbound"
directive="query-sync">
  <oid>
    <Ind>RuleML-2012-ECAI</Ind>
  </oid>
  <protocol>
    <Ind>esb</Ind>
  </protocol>
  <sender>
    <Ind>User</Ind>
  </sender>
  <content>
    <Atom>
      <Rel>getTracks</Rel>
      <Var>Track</Var>
    </Atom>
  </content>
</Message>
Using these messages agents can interchange events (e.g. queries and answers) as well as complete rule bases. Agents engaged in this communication can keep track of their conversation using

- oid - Conversation ID.
- Protocol - Message passing and coordination protocol.
- Directive - This attribute corresponds to the pragmatic instruction (characterizing the meaning of the message).

The Reaction RuleML translator services are part of the configuration of the Enterprise Service Bus (ESB) which we shall discuss independently. However, it is important to know for now that the ESB is able translate the incoming and outgoing messages through deployed rule engines on the inbound and outbound links of the ESB. Incoming Reaction RuleML messages are translated into platform specific rulebases which can be executed by the rule engine (in our case Prova) and outgoing rulebases are translated into Reaction RuleML.

### 6.5 Performatives

Rule Responder is a distributed rule system, where each rule agent can run a different rule engine having its own proprietary syntax to access different
sources of local data. Usually these distributed agents connect and communicate with each other based on a common rule interchange language, which carries pragmatic performatives. These performatives can be used by the receiver agents to understand the pragmatic context of the message.

As discussed earlier, the main language constructs of messaging reaction rules are:

- **sendMsg** predicate: These are used to send messages.

- **rcvMsg** predicate: Rules which react to inbound messages.

- **rcvMsg** or **rcvMult**: To receive one or more context-dependent multiple inbound messages.

```
sendMsg(XID, Protocol, Agent, Performative, Payload |Context)
rcvMsg(XID, Protocol, From, Performative, Payload|Context)
rcvMult(XID, Protocol, From, Performative, Payload|Context)
```

Here,

- **XID** - Conversation Identifier of the conversation.


- **Performative** - Pragmatic envelope for the message content. A standard nomenclature of performatives is FIPA Agents Communication Language (ACL).
- Payload - Message content sent in the message envelope. The payload of the incoming messages is interpreted with respect to the local state i.e (conversation id).

Prova agents can interchange information, rules and queries/answers in agent conversations, including information about the semantics and pragmatics of the interchanged information [26].

6.6 Agent Communication Protocols

Our system implements different communication protocols which our agents can utilize. The protocols vary by the number of steps involved in communication. We attempt to follow message patterns similar to the Web Service Description Language (WSDL). For example, there can be in-only, request-response, request-response-acknowledge protocols, as well as workflow protocols. Our system primarily focuses on request-response protocol. The different protocols are explained below [34],

- In-Only: Agent 1 sends a message to Agent 2, which then executes a performative. For example, the OA sends a performative to a PA to either assert or retract a clause.

- Request-Response: It starts like In-Only protocol but the Agent 2 sends a response. For example, the OA sends a query to a PA and PA solves the query before sending solution back to OA.
- Request-Response-Acknowledge: The request-response-acknowledge communication protocol starts like request-response but Agent 1 sends an acknowledgement message back to Agent 2. For example, the OA sends query to PA, the PA solves the query and sends the answer back to the OA, and the OA sends an acknowledgement about having received the query’s answer to the PA.

- Workflow: A workflow communication protocol generalizes the above sequential protocols by allowing an arbitrary composition of agent messages from sequential, parallel, split, merge, conditional elements, and loops. For example, the EA sends a query to the OA, the OA decomposes and delegates the subqueries to two PAs, the PAs solve their subqueries sending back the answers to the OA, and the OA integrates the answers and sends the final answer back to the EA.
6.7 System Evaluation

In order to evaluate the system, different queries/scenarios can lead to differing levels of performance. SymposiumPlanner is built on top of the Rule Responder which has a good overall performance. When the user makes a change in the query index in the External Agent (interface for human user interaction). The change is almost instantaneous as it is using javascript in background to change the query based on user choice.

The SymposiumPlanner has three main agents which are interacting to execute queries according to user queries. The Personal Agent (PA) includes interaction with local knowledgebases as well as the global knowledgebases which are part of the Organizational Agents (OA) which include a set of rules and predicates to decipher and help delegate tasks to the PAs using a responsibility assignment matrix while the EA is able to access PA and OA knowledgebases using a web browser in our case for interaction.

The OA performance can be affected by time to translate messages between endpoints (mule service points) using disparate rule engines. Although the system uses reaction ruleml for systemwide interaction but different rule engines are employed. At the beginning of the search for the suitable PA, the OA is given an initial 5 min timeout to carry out the initial execution as well as searching for the PA most suitable for answering the entered query.

```java
if (temp.contains("RuleML-2012"))
{

```
Hence, when the system receives a query from the user, it can lead to two situations.

1. The OA is able to complete the execution in time and delegates correctly as well as responds correctly.

2. The OA fails to complete the execution in time and is not able to locate the correct PA and goes into a long waiting state.

**OpenRuleBench Prova Performance Analysis** SymposiumPlanner implements global knowledge bases using Prova. Prova is also used to implement the logic for deciphering incoming queries and delegating the queries to the Personal agents based on their respective roles defined in the role assignment matrix which is an OWL lite ontology.

OpenRuleBench [31] finding regarding Prova are summarized in the listing [2] below:

- Prova fails to delete and retrieve all inserted facts.

- It shows lack of performance for data loading.

- It shows lack of performance for computing time.
These limitations in Prova 2 (used in the development of this system) cause delay in receiving and execution of facts.

**Servlet Container Monitoring** At the heart of the SymposiumPlanner, are servlets that represent the Personal Agents. These servlets were analyzed using PSI Probe (http://code.google.com/p/psi-probe/). PSI Probe is an intended replacement and extension of Tomcat Manager. It has a rich list of features to offer which are as follows:

- Requests: Monitor traffic in real-time, even on a per-application basis.
- Sessions: Browse/search attributes, view last IP, expire, estimate size.
- JSP: Browse, view source, compile.
- Data Sources: View pool usage, execute queries.
- Logs: View contents, download, change levels at runtime.
- Threads: View execution stack, kill.
- Connectors: Status, usage charts.
- Cluster: Status, usage charts.
- JVM: Memory usage charts, advise GC
- Java Service Wrapper: Restart JVM.
- System: CPU usage, memory usage, swap file usage.
The servlet performance is directly proportional to the amount of data in that PA’s profile. PSI probe was used to monitor the publicity chair PA. The statistics show that specific queries can affect system performance. The sponsorship query has a list of facts and rules to support the Symposium-Planner in giving a response and that shows in system performance analysis as shown in 6.3 the servlet response time reaches its peak when it is asked something about sponsoring.

In the same servlet a query like mediafeedresources will take up less resources than sponsoring and the system response times vary as shown in 6.4 from min time to max time.

**System Improvements**

The system performance can be improved by taking following steps:

- Prova 3 instead of Prova 2.
  Prova 3 includes many improvements to optimize intra-JVM [21] message reactions aimed at accelerating intra-agent message passing. This can enhance performance of the system as the incoming reaction message is matched to rules inside the OA which is implemented using Prova.

- Mule 3 ESB.

- Use of local knowledge bases implemented in Prova to hold information increases the speed of system responses manifold.
Figure 6.3: Publicity Chair Servlet
- Use of Tomcat 7 as it has better garbage collection than earlier versions which can cause memory leaks which affects performance.
Chapter 7

Conclusions

In this thesis, we have implemented SymposiumPlanner 2012 which is an instantiation of Rule Responder that attempts to model the organizing committee of a conference in the virtual world as well as an organization for the first time. The illustrative examples of the system development have been discussed in previous sections.

7.1 Contributions

- Our system uses and improves on the design principles in the Rule Responder to improve the implementation of SymposiumPlanner. i.e. instead of Semantic Dog Food, locally available information was used while the structure was left intact so that it can be used in future if needed.
- Our system allows different rule engines to execute global and local rulebases.

- Several new PAs were introduced in the system according to the requirements of the RuleML Symposium held in conjunction with ECAI 2012.

- Facts and rules for the new PAs were added to the system in POSL based profiles and improved naming conventions were applied.

- RuleML (http://ruleml.org/) is an organization structure was introduced into the system. The technical groups were mapped into the system to form Personal Agents (PAs) with their knowledge bases.

### 7.2 Future Work

- The new RuleML-Structure sub-OA was introduced into the SymposiumPlanner 2012 which can be further developed to have a more rules and facts in the knowledge bases.

- The Mule ESB 3.3 can be integrated into the system.

- Prova 3 can be implemented overcoming the JVM intra agent communication limitations in Prova 2.
Bibliography


Appendix A

System Resources

The system resources can be accessed on-line using the links displayed under the respective headings.

A.1 User Client


A.2 Prova Knowledge Bases

A.2.1 Super-Organizational Agent (RuleML-2012 RuleML Structure)

A.2.2 Sub-Organizational Agent (RuleML-2012)


A.2.3 Sub-Organizational Agent (RuleML Structure)


A.3 Responsibility Assignment Matrix


A.3.1 Sub-Organizational Agent Responsibility Assignment Matrix

- Sub-Organizational Agent (RuleML-2012)

- Sub-Organizational Agent (RuleML Structure)

A.4 Personal Agent Profiles

A.4.1 General Chair

A.4.2 Publicity Chair


A.4.3 Program Chair


A.4.4 Doctoral Symposium Chair


A.4.5 Int’l Rule Challenge Steering Committee Chair


A.4.6 Int’l Rule Challenge Chair


A.4.7 Local Chair

A.4.8 Metadata and Social Media Chair


A.4.9 Steering Chair


A.4.10 RuleML Technical Groups

Glossary (if any)

RuleML - RuleML Markup Language
OO RuleML - Object Oriented Rule Markup Language
FOAF - the Friend Of A Friend project
RDF - Resource Description Framework
RDFS - Resource Description Framework Schemas
POSL - Positional-Slotted Language
XSLT - The Extensible Stylesheet Language Transformations
XML - Extensible Markup Language
jDREW - java Deductive Reasoning Engine for the Web
OO jDREW - Object Oriented java Deductive Reasoning Engine for the Web
WWW - World Wide Web
URI - Universal Resource Identifier
HTTP - Hypertext Transfer Protocol
HTML - Hypertext Markup Language
W3C - World Wide Web Consortium
SP - SymposiumPlanner
OA - Organizational Agent
Super-OA - Super Organizational Agent
Sub-OA - Sub Organizational Agent
PA - Personal Agent
EA - External Agent
ESB - Enterprise Service Bus
TD - Top Down
BU - Bottom Up
Prolog - PROgramming in LOGic
OWL - Web Ontology Language
SOAP - Simple Object Access Protocol
IRI - Internationalized Resource Identifier
JMS - Java Message Service
UMO - Universal Message Object
XSL - EXtensible Stylesheet Language
Prova - Prolog+Java
OSGi - Open Services Gateway initiative framework
JVM - Java Virtual Machine
Vita

Candidate’s full name:
University attended (with dates and degrees obtained):
Publications:
Conference Presentations: