EnviroPlanner: Design and Implementation of a Distributed Environmental Querying System in Rule Responder

by

Sujan Chandra Saha

Bachelor of Information Technology, Multimedia University, Melaka, Malaysia, 2008

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

Masters of Computer Science

In the Graduate Academic Unit of Faculty of Computer Science

Supervisor(s): Harold Boley, PhD, Computer Science
Gerhard Dueck, PhD, Computer Science
Examine Board: Przemyslaw Poche, PhD, Computer Science, Chair
Virendra Bhavasar, PhD, Computer Science
Donglei Du, PhD, Business Administration

This thesis is accepted by the Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK
September, 2013
©Sujan Chandra Saha, 2013
Abstract

Environmental information querying can be cumbersome and time-consuming as users sometimes need to go through multiple Web pages for finding answers to specific questions. As a prototype modeling a multi-agent Virtual Environmental Organization (VEO), EnviroPlanner is developed to allow users to retrieve and deduce environmental information via problem-oriented question answering. In this thesis, we focus on the design and implementation of the EnviroPlanner VEO model that is supported by rule and ontology knowledge. This formalized knowledge allows EnviroPlanner’s semi-automated agents to assist human experts in environmental question answering. We realized EnviroPlanner for distributed querying in the Rule Responder framework that consists of three kinds of agents: External Agent (EA), Organizational Agent (OA), and Personal Agents (PAs). The EA is the single point of entry that allows users to pose queries to the system, employing a Web interface coupled to an HTTP port to which requests are sent. EnviroPlanner consists of an extension and an instantiation of the Rule Responder framework similar in the communication architecture to the SymposiumPlanner-2011/2012 instanti-
ations but in a very different knowledge domain. The SymposiumPlanner systems since 2011 have used two Sub-Organizational Agents (Sub-OAs) for question answering. We extended the framework to provide architectural flexibility to developers so they can add as many Sub-OAs as needed, e.g. for answering user queries about different locations or regions. The architecture designed for environmental query answering has been implemented and the EnviroPlanner prototype has been evaluated with respect to efficiency and overall performance. In addition, the EnviroPlanner prototype has been deployed on the official Rule Responder website.
Dedication

This thesis is dedicated to my parents.
Acknowledgements

I would like to express my sincere gratitude to my advisors, Dr. Gerhard Dueck and Dr. Harold Boley, for their motivation, guidance, and support during my thesis. Continuous discussions and valuable input from them made it possible for me to complete this thesis.

I would also like to thank Zhili Zhao for his wonderful technical support, suggestions, and feedback regarding Rule Responder from the very beginning of my thesis. By sharing his Rule Responder expertise, he enabled the EnviroPlanner project. Moreover, I would like to thank Chaudhry Usman Ali for the fruitful exchange in our initial exploration of Rule Responder and SymposiumPlanner.

I would like to express my gratitude to all of my friends and family for their continuous support.
# Table of Contents

Abstract ii

Dedication iv

Acknowledgments v

Table of Contents x

List of Tables xi

List of Figures xiii

Abbreviations xiv

1 Introduction 1
   1.1 Thesis Objectives and Methodology 3
   1.2 Thesis Organization 5

2 Background 7
   2.1 The Semantic Web 9
      2.1.1 Resource Description Framework 10
2.1.2 RDF Schema
2.1.3 Web Ontology Language
2.1.4 Inference Process
2.1.5 Friend of a Friend (FOAF)

2.2 Virtual Environmental Organization
2.2.1 Intelligent Agents
2.2.2 Agent Communication
  2.2.2.1 Agent Communication Language
  2.2.2.2 Agent Communication Protocol
2.2.3 Rule Responder
  2.2.3.1 Rule Responder Architecture
  2.2.3.2 Communication Middleware

3 Rule Languages and Tools
3.1 Rules
3.2 Rule Languages
  3.2.1 Rule Markup Language
  3.2.2 Reaction RuleML
  3.2.3 Positional-Slotted Language (POSL)
3.3 Semantic Web Tools and Rule Engines
  3.3.1 Prova
    3.3.1.1 General Prova Features
    3.3.1.2 Reactive messaging
3.3.1.3 Sending Messages ......................... 34
3.3.1.4 Receiving Messages ....................... 37
3.3.2 OO jDREW .................................. 39
3.3.3 XSLT ...................................... 41

4 EnviroPlanner Architecture ................. 43
  4.1 Knowledge Representation .................. 45
  4.2 Distributed System .......................... 46
    4.2.1 Distributed System Architecture .......... 47
    4.2.2 Distributed Rule System .................. 49
  4.3 Organizational Agents ....................... 50
    4.3.1 Super-Organizational Agent ............... 52
    4.3.2 Sub-Organizational Agent ................ 53
    4.3.3 Responsibility Assignment Matrix .......... 55
    4.3.4 Performatives ............................ 58
  4.4 Personal Agents .............................. 59
    4.4.1 Translation Between EnviroPlanner Agents .. 60
    4.4.2 Query Answering for Personal Agents ....... 64
  4.5 External Agent ................................ 66
  4.6 Enterprise Service Bus ....................... 67

5 Local and Global Knowledge Bases .......... 77
  5.1 Fact- and Rule-Based Scripts for PAs ........ 77
    5.1.1 Fact and Rule Processing in a Profile .... 89
A.2 Experts' Profile Examples ........................................ 135
  A.2.1 Fredericton Region Air ..................................... 135
  A.2.2 Fredericton Region Energy ................................. 146
  A.2.3 Fredericton Region Soil ................................... 148
  A.2.4 Fredericton Region Water ................................. 150
A.3 EnviroPlanner Role Assignment Matrix ....................... 153

Glossary 177

Vita
List of Tables

6.1 Running Time in Seconds ......................... 110
6.2 Comparison of Performance ..................... 113
6.3 Response Time on Different Versions of Mule ........ 117
6.4 Statistical Significance of Size of KBs .......... 118
List of Figures

2.1 Semantic Web Layers ........................................ 11
2.2 RDF Triples ..................................................... 12
2.3 RDF Graph ....................................................... 13
2.4 Overview of the FIPA ACL Specifications .................. 20
2.5 Rule Responder Architecture ................................. 22

3.1 Process Flow of XSLT .......................................... 42

4.1 EnviroPlanner Architecture ................................. 44
4.2 Architecture of a Simple Distributed System .............. 46
4.3 Star Topology .................................................... 48
4.4 Connection of Sub-OAs to the Super-OA .................. 54
4.5 External Agent of EnviroPlanner ............................ 67
4.6 Mule Enterprise Service Bus ................................. 68
4.7 Transport Mechanism of Mule ............................... 69
4.8 Use of Mule in EnviroPlanner ............................... 71

6.1 EnviroPlanner User Interface ............................... 103
6.2 Response Time of the 'Emission Neutralizer' Facts ........ 109
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>Response Time After Having Doubled the Facts</td>
<td>109</td>
</tr>
<tr>
<td>6.4</td>
<td>Response Time After Having Tripled the Facts</td>
<td>109</td>
</tr>
<tr>
<td>6.5</td>
<td>Response Time Comparison Among Super-OA and Sub-OA levels</td>
<td>112</td>
</tr>
<tr>
<td>6.6</td>
<td>Analysis of EnviroPlanner</td>
<td>114</td>
</tr>
<tr>
<td>6.7</td>
<td>Analysis of SP-2012</td>
<td>114</td>
</tr>
</tbody>
</table>
**List of Abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>External Agent</td>
</tr>
<tr>
<td>ENB</td>
<td>Environment New Brunswick</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>VEO</td>
<td>Virtual Environmental Organization</td>
</tr>
<tr>
<td>FOAF</td>
<td>Friend Of A Friend</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IRI</td>
<td>Internationalized Resource Identifier</td>
</tr>
<tr>
<td>jDREW</td>
<td>java Deductive Reasoning Engine for the Web</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>KB</td>
<td>Knowledge Base</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent System</td>
</tr>
<tr>
<td>OA</td>
<td>Organizational Agent</td>
</tr>
<tr>
<td>OSGi</td>
<td>Open Services Gateway Initiative Framework</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PA</td>
<td>Personal Agent</td>
</tr>
<tr>
<td>Prolog</td>
<td>PROgramming in LOGic</td>
</tr>
<tr>
<td>Prova</td>
<td>Prolog+Java</td>
</tr>
<tr>
<td>POSL</td>
<td>Positional Slotted Language</td>
</tr>
<tr>
<td>RAM</td>
<td>Responsibility Assignment Matrix</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>Resource Description Framework Schema</td>
</tr>
<tr>
<td>RuleML</td>
<td>Rule Markup Language</td>
</tr>
<tr>
<td>SPARQL</td>
<td>Simple Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Sub-OA</td>
<td>Sub Organizational Agent</td>
</tr>
<tr>
<td>Super-OA</td>
<td>Super Organizational Agent</td>
</tr>
<tr>
<td>UMO</td>
<td>Universal Message Object</td>
</tr>
<tr>
<td>VO</td>
<td>Virtual Organization</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XSL</td>
<td>Extensible Stylesheet Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language Transform</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Disclaimer: The software that has been developed in this thesis is a prototype not intended to support environmental decision-making nor to provide the required knowledge in areas such as physics, chemistry, biology, or engineering. The collected examples are given as a use case for Computer Science methods, techniques, and tools. Except for the Mactaquac Dam example, all names and data in the examples are fictitious.

A large amount and variety of data is being created and maintained all over the Internet, making relevant information hard to find. Therefore, to query specific environmental information from this large amount of data, the importance of problem-oriented question answering is now being realized. During the literature and background research of this thesis, it was observed
that existing environmental websites are not able to answer many kinds of queries appropriately. To get answers to specific questions, users often still have to find and contact relevant (government or non-government) organizations. Moreover, currently available environmental information on the Web is mainly in the form of human-presentable documents (e.g., in HTML or Word) rather than machine-formalized knowledge (e.g., in XML or RDF). Thus, the information cannot easily be processed in an automatic manner.

Typically, users pose queries to search engines on the Web. These queries may retrieve interesting documents, but search engines are still not usually performing (environmental) question answering [45] due to the lack of machine formalized (environmental) knowledge and inference engines. A knowledge-based Multi-Agent System (MAS) [51] can help with this problem through an interconnected distributed architecture. In particular, a Virtual Environmental Organization (VEO) can be supported by rules and ontologies [53] that allow semi-automated agents to assist human experts in question answering; these agents communicate with other agents for distributed query processing.

This thesis proposes EnviroPlanner as a distributed system for environmental querying, which could be developed into a VEO supporting government and non-government organizations in problem-oriented query answering. This can also be seen as an application of environmental Linked Open Data (LOD).
[63], a term used to describe the practice of exposing, sharing, and connecting pieces of data and knowledge on the Semantic Web using Uniform Resource Identifiers (URIs) and the Resource Description Framework (RDF). These environmental open data can be used as the basis for the rules and ontologies provided by the experts of a VEO, such as eGovernment-augmented municipal or provincial environmental departments. By using a further developed EnviroPlanner, citizens will be able to inquire their VEO about environmental issues, such as air, soil, water, and energy, and answers will be searched for them via location- and issue-specific query delegation.

1.1 Thesis Objectives and Methodology

The main objective of this thesis is in the following goal of semantic question answering where appropriate to design, implement, and evaluate a distributed environmental information system, which acts as a semi-automated environmental query answerer and supports decision making. This is achieved by the following methodology:

1. The system provides Personal Agents (PAs) which can assist a VEO to answer user queries. The PAs use Knowledge Bases (KBs) that encode knowledge of human experts in the VEO. Each KB is stored locally as an expert profile of a PA. Environmental facts and rules are included in the respective PAs’ profiles. Such an environmental knowledge profile is interpreted by the corresponding expert’s PA to answer user queries
in the experts’ stead; these queries are delegated by the Sub-OA of the VEO to the PA that is responsible for answering the queries.

2. EnviroPlanner builds on SymposiumPlanner-2011 [63], an instantiation of the Rule Responder framework [54] which is similar in the communication architecture but not in the knowledge domain. To enable greater architectural flexibility, we extended the functionality of SymposiumPlanner-2011, allowing to add as many Sub-OAs as needed. In our system, we deploy three Sub-OAs, one for Fredericton, one for Moncton, and another one for Saint John. Each Sub-OA is designed to deal with air, soil, water, and energy issues of its region. In addition, on top of these Sub-OAs, a Super-OA is designed to delegate region-specific queries to the appropriate Sub-OA.

3. Rule Responder allows different rule engines to execute global and local rulebases. EnviroPlanner uses the Prova rule engine to execute global knowledge bases and OO jDREW to execute local knowledge bases.

4. A lightweight ontology is created in the Web Ontology Language (OWL) realizing the Responsibility Assignment Matrix (RAM), which is used to capture query keywords that describe roles and responsibilities of the Sub-OAs and PAs in the semantic question-answering system. The Super-OA acts as a dispatcher to redirect requests sent by a user via the EA to one of the three Sub-OAs. Sub-OAs keep on filtering and delegating incoming queries to the responsible regional members which
are implemented as distributed rule-based PAs.

5. Rule Responder, hence EnviroPlanner, allows facts/rules, queries, and their answers to be transmitted over an Enterprise Service Bus (ESB). An ESB creates a communication endpoint at each Personal and Organizational Agent of EnviroPlanner. In particular, the open-source lightweight event-driven Mule ESB is being used as the communication middleware.

6. Evaluation of the EnviroPlanner Prototype:
   - Evaluate the completeness and consistency of the KBs
   - Evaluate the performance and scalability of the system
   - Evaluate the functional efficiency of EnviroPlanner as a Rule Responder instantiation.

1.2 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2 presents the concepts and framework that are at the foundation of EnviroPlanner. Some of the important components of the Semantic Web and the basic framework of Rule Responder are briefly described as required for EnviroPlanner. Chapter 3 provides the details of rule languages and rule engines that have been used in this thesis. How rule languages are used...
by the rule engines to interchange and share knowledge through the system is described here. In Chapter 4, the EnviroPlanner system is described in detail. An approach aimed at achieving the goal of this thesis, which is developing a semantic question-answering system for querying in the environmental domain, is presented. It includes the description of the system architecture, the functionality of each agent, and the implementation of the RAM algorithm that is used to map the query to the appropriate PA profile of a human expert. Chapter 5 describes the details of environmental KB rule representation. It discusses the PA profiles that contain the facts and rules for answering environmental queries. Experimental results of some illustrative examples are given here. In Chapter 6, we describe the user interface, query description in controlled English, and the evaluation process, performance analysis, and scalability of EnviroPlanner. Finally, the conclusion and some proposed future work are presented in Chapter 7.
Chapter 2

Background

The availability of consistent, relevant, and up-to-date information, especially environmental information, is one of the prerequisites for rational and cost-effective environmental information processes. Because of the lack of machine-formalized knowledge (e.g. in XML or RDF), users often do not get the results they are looking for on the Web. For example, if a user poses the following query on a search engine, the user might get thousands of results instead of getting the expected result.

How many air pollutants (acids) were released by the ABC chemical company in 2013, and what chemical bases can be used to neutralize those pollutants?

Various federal and provincial environmental websites of Canada have been researched, but these websites are not able to answer many kinds of specific problem-oriented queries with their traditional Web architecture. To obtain
the appropriate results, users either need to go through multiple Web pages or have to find and contact relevant environmental agencies, which can be time-consuming and inconvenient.

Moreover, in order to benefit from the rapid technological growth, it is very important to have mechanisms that will analyze and process the information easily and quickly. Users want access to an intelligent and easy-to-use system to get the right information timely. Despite the increasing effectiveness and sophistication of information and communication technology that is enabling the spread of interconnected distributed VOs, there has been very little research on exploring the effectiveness of a Virtual Environmental Organization (VEO).

Rule Responder is an extended Semantic Web framework for distributed rulebases, structured by ontologies. Rules and ontologies are the main knowledge representations on the Semantic Web [43]. They can be used by distributed semi-automated agents. A semi-automated agent, which is a rule/ontology-based agent in a VEO that uses rules/ontologies to describe the decision logic of an expert in the VEO, could be a key factor in realizing the vision of the Semantic Web. Semi-automated agents might play an important role in many of the technologies proposed by the World Wide Web Consortium (W3C)

\[^{1}\text{http://www.w3.org/}\]
cially when environmental information and resources are spatially distributed across interconnected networks. In fact, to drive the evolution of the current Web, we need semantic agent systems allowing users to search, find, filter, share, combine, and transfer the information easily and quickly. The Semantic Web provides an infrastructure that has the potential to transform the Web into a global knowledge medium [57]. Rule Responder provides the necessary infrastructure for creating a VO as a Multi-Agent System (MAS) [54]. Various Rule Responder instantiations, such as PatientSupporter [33], WellnessRules [29], and SymposiumPlanner [64] have been developed as VOs but no Rule Responder VEO has been developed yet to address environmental queries.

2.1 The Semantic Web

In the Semantic Web, data can be linked in such a way that machines can understand and process these data easily. It is a collaborative effort led by the international standards body W3C [5]. According to the W3C, Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data [15]. These webs of linked data are empowered by Semantic Web technologies such as OWL (Web Ontology Language), RDFS (Resource Description Framework Schema), RDF (Resource Description Framework), SPARQL (SPARQL Protocol and RDF
Query Language), and RIF (Rule Interchange Format). Rules and ontologies are the main knowledge representations on the Semantic Web. The Semantic Web stack, also known as the Semantic Web Cake or Semantic Web Layer Cake, illustrates the architecture of the Semantic Web, which is given in Figure 2.1.

The Semantic Web components that are related to this thesis are discussed in the following.

2.1.1 Resource Description Framework

Resource Description Framework (RDF) is a XML-based language for representing information about Web resources. RDF information can be processed by applications rather than only being displayed to users. RDF provides a common framework for expressing information so it can be exchanged between applications without loss of meaning [13]. RDF is written in XML, and it uses a triple expression:

- Subject: Represents the resource
- Predicate: It is the relationship between the subject and object
- Object: Represents the object in the relationship

If we visualize the RDF triples as a directed labeled graph, as in Figure 2.2, subjects and objects can be represented as nodes, and predicates as arcs.
User Interface & Applications

Unifying Logic

Ontology: OWL

Rule: RIF

Query: SPARQL

RDFS

Data interchange: RDF

XML

URI/IRI

Proof

Trust

Crypto

Figure 2.1: Semantic Web Layers [52]
RDF identifies resources using the Uniform Resource Identifiers (URIs) [17],
which is a string of characters used to identify a name or a resource, and
describes resources with properties and property values. One of its main ben­
efits is mapping information directly and unambiguously to a decentralized
model that indicates which bits of data are the semantics of the application,
and which bits are just syntactic [16]. The following example is an RDF
document that describes the resource for an expert.

```xml
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:person="http://example.ca/EnviroPlanner#">
    <rdf:Description rdf:about="http://example.ca/EnviroPlanner/experts">
        <person:title>Dr.</person:title>
        <person:name>Donald Smith</person:name>
        <person:areaOfInterest>Energy</person:areaOfInterest>
    </rdf:Description>
</rdf:RDF>
```

Figure 2.2: RDF Triples
In the above example, the first line is the XML declaration. The XML declaration is followed by the root element of RDF documents, `<rdf:RDF>`. The `<rdf:Description>` element contains the description of the resource identified by the rdf:about attribute; `http://example.org/EnviroPlanner/experts`. The elements `<person:title>`, `<person:name>`, and `<person:areaOfInterest>` are properties of the resource; Dr., Donald Smith, and Energy are the values of these properties, respectively. Figure 2.3 shows the graph model of the above RDF example.

![RDF Graph](image)

**Figure 2.3: RDF Graph**

### 2.1.2 RDF Schema

RDF Schema (RDFS) extends the RDF vocabulary that allows describing the taxonomies of classes and properties of a domain [11]. The main intention of RDFS is to structure the RDF resources. It is used for declaring basic classes and types when describing the terms used in RDF; thus RDFS is used to determine
the characteristics of other resources [16]. RDF resources can be divided into groups called classes. Classes are described using properties of the resources and identified by the URIs. The members of a class are instances of the classes, which are stated using the \texttt{rdf:type} property [16].

The re-usability of the RDFS reduces the ambiguity of ontologies, and increases the equality and clarity of the ontologies because existing ontologies are usually tested. For instance, our EnviroPlanner reuses some patterns of the existing ontologies of \textit{SymposiumPlanner-2012} [64].

### 2.1.3 Web Ontology Language

Web Ontology Language (OWL) is one of the knowledge representation languages that is used for authoring ontologies by providing additional vocabularies along with a formal semantics [25]. According to the W3C [15], "OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans". Ontologies, in any domain, provide the reference models that can be used to establish communication and sharing of knowledge between agents and human experts [40]. OWL 1 has three sublanguages (a similar layering exists for OWL 2):

- **OWL Lite**: Simplest form of OWL
- **OWL DL**: Based on Description Logics
- **OWL Full**: Most expressive version of the OWL
For our system, we adopted OWL Lite to describe the Responsibility Assignment Matrix (RAM). This ontology is used by the Organizational Agents to describe the responsibilities of the Personal Agents.

2.1.4 Inference Process

Inference is a process of drawing a conclusion from a set of given rules. According to [5], the inference process generates new relationships based on the given data and additional information in the form of a vocabulary, e.g. a set of rules. Additional information can be defined in a set of rules using knowledge representation techniques. Inference process derives the answers after executing the logical flow of a knowledge base [21] where information can be stored as a repository that provides a means for information to be organized, searched, and utilized [59]. We used the Prova and Object Oriented Java Deductive Reasoning Engine (OO jDREW) [37] rule engines for top-down inference process that retrieves the information from the KBs after the completion of inferences.

2.1.5 Friend of a Friend (FOAF)

FOAF is a machine-readable vocabulary that is used to describe people, the links between them, their activities and relations to other people, objects, and activities [4]. FOAF is built using decentralized Semantic Web technology, and designed to allow data integration across a variety of applications, websites, and services [34]. Like other RDF vocabularies, FOAF vocabularies can be combined with other vocabularies, which allows a very rich set of meta-data [41]. Regardless of the data
form, FOAF data can be linked through the Semantic Web. The person class is one of the core classes of the FOAF vocabulary. We created experts' FOAF-like profiles to get the relevant personal details of a person from EnviroPlanner. The following FOAF profile states that James Anderson is the name of a person, his title is Mr., and his e-mail address is jAnderson@enb.com. He is working as an air quality expert and, he is interested in the environment. His interest and job are resources, which means that each of these can be described in another RDF vocabulary format.

```xml
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"

<foaf:Person rdf:ID="James">
   <foaf:title>Mr.</foaf:title>
   <foaf:name>James Anderson</foaf:name>
   <foaf:givenname>James</foaf:givenname>
   <foaf:family_name>Anderson</foaf:family_name>
   <foaf:mbox <mailto:jAnderson@enb.com>
   <foaf:interest rdf:resource="Environment"/>
   <foaf:job rdf:resource="Air Quality expert"/>
</foaf:Person>
</rdf:RDF>
```
2.2 Virtual Environmental Organization

A Virtual Environmental Organization (VEO) is a multi-agent knowledge-based mechanism for enabling coordination among various heterogeneous agents/services, where different rules and ontologies can be executed by different agents. Individuals and institutions/organizations can use VEOs to coordinate resources and services across institutional boundaries [39]. A VEO allows an organization to describe its structure, behavior, goals, norms, and environment through its intelligent agents.

2.2.1 Intelligent Agents

According to Woodridge and Jennings [62], an agent is an an entity whose behavior can be predicted by the method of attributing beliefs, desires, and rational acumen. Agents often impute different human characteristics to it, such as knowledge, and obligation. An agent is an encapsulated computer system that is made up of an architecture and a program. Software-based agents have the following properties [60]:

- Autonomy: Agents can operate without the direct intervention of humans or other agents.

- Social ability: Agents interact with other agents via some kind of agent-communication language. (Our system uses RuleML as an agent-communication language).

- Pro-activeness: Agents are able to exhibit goal-oriented behavior by taking the initiative.
When agents interact with other agents, the intended result is a distributed network of intelligent cooperation, which leads to the delivery of useful services. A number of tasks can be accomplished with agents and the Semantic Web [42]. For example:

- An agent can turn documents into formal Semantic Web-based knowledge.
- A multi-agent system can operate in its community or in a distributed environment to build and maintain Linked Data sets.

### 2.2.2 Agent Communication

In a Virtual Organization, intelligent agents need to interact with other agents so that they can share or exchange information in a way similar to humans by using different Agent Communication Language (ACL), which is described below.

#### 2.2.2.1 Agent Communication Language

Specifications for various languages have been proposed for agent communication, such as Knowledge Query and Manipulation Languages (KQML) [6] and Intelligent Physical Agents (FIPA) [3].

**Knowledge Query Management Language**

The first significant ACL was the Knowledge Query Management Language (KQML) [59], which includes many primitives, assertives, or directives that agents use to ask queries, tell facts, or find other agents. The KQML semantics presupposes a virtual knowledge base for each agent. Telling a fact corresponds to reporting on
that knowledge base; querying corresponds to the sending agent’s attempt to ex-
tract something from the receiving agent’s knowledge base [59]. But the challenge 
of developing a capability for agents to initiate a communication on their own still 
remains.

**FIPA Specifications**

The recent evolution of Agent Communication Languages has been guided by 
the standard specifications proposed by the Foundation for Intelligent Physical 
Agents (FIPA). These specifications explicitly outlined the interaction protocols 
not covered by the KQML. Figure 2.4 shows that FIPA agent communication 
specifications deal with Agent Communication Language (ACL) messages, message 
exchange interaction protocols, speech act theory-based communicative acts, and 
content language representations. The set of speech acts forms the basic set of 
message types exchanged between agents. The interaction protocols provided by 
FIPA include task allocation, negotiation, and active directories.

Instead of providing a single mandatory content language, FIPA provides a set 
of reference content languages, such as Knowledge Interchange Format (KIF) and 
Resource Description Framework (RDF). FIPA has made significant progress in 
advancing interoperability between agents, and laid down the basis for open source 
implementations to bring agent technology into world deployment.

**2.2.2.2 Agent Communication Protocol**

In order for an agent to exchange queries, rulebases, or query-answers with other 
agents in a secure manner, the agent communication mechanism needs a commu-
A communication protocol can be defined as a set of rules that governs the communication between various components of a system [49]. There are various types of protocols, such as request-response, and request-response-acknowledge protocols. The protocols vary by the number of steps involved in an agent communication. EnviroPlanner focuses on the request-response protocol.

In a request-response protocol, when an agent sends a request to another agent, the second agent executes its performative and sends a response to the first agent. For example, when an OA of our system sends a query request to a Personal Agent, it executes its performative to solve the query and sends the solution back to the OA. If there is no solution available, it sends a response back to the OA indicating that it cannot find a solution.

![Diagram of Abstract Architecture](image)

**Figure 2.4: Overview of the FIPA ACL Specifications [58]**
2.2.3 Rule Responder

Rule Responder [31] is widely used Multi Agent Systems (MAS) for a VO that extends the Semantic Web towards a Pragmatic Web infrastructure for collaborative human-computer networks. It provides an effective methodology and an efficient infrastructure for interchange and reuses knowledge that can play an important role in drawing a conclusion from existing knowledge [18]. As a distributed framework, all of its instantiations, e.g. PatientSupporter [33], WellnessRules2 [18], SymposiumPlanner [64], have been implemented as a Web-based service architecture where Rule Responder’s agents communicate with other agents to achieve the goals of an organization.

2.2.3.1 Rule Responder Architecture

Figure 2.5 shows the top-level architectural view of Rule Responder which consists of three main types of agents, and it has three core parts [54] that are given below:

- Interchange format: It establishes the platform-independent interchange of rules and events between agents/services

- Communication middleware: It transports the queries, answers, and information between agents

- Rule engines and execution environments: They permit distributed heterogeneous information processing

As a multi-agent system, the following three main agents of Rule Responder play an important role in a VEO:
• Personal Agent (PA): Derives information from its local KB

• Organizational Agent (OA): Represents an organization as a whole to achieve its goals

• External Agent (EA): Works on a Web interface level

2.2.3.2 Communication Middleware

A communication middleware framework provides an environment where multiple agents/applications can communicate with each other and exchange information among themselves. This environment can be either [35]:

• Hub-and-spoke model: A centralized architecture in which all data exchange is processed by a hub. This model is used by the Enterprise Application
Integration (EAI) products, where applications are designed to run independently with limited or no interaction between applications.

- **Bus model**: A distributed architecture, in which ESB functionalities can be implemented by several physically separated functions. Because of its use of open standards, the bus model is widely used by distributed ESB products.

To seamlessly handle message-based interactions between the agents/services, we used the Mule ESB [7], a communication middleware that uses disparate complex event processing (CEP) technologies, transports, and protocols. Mule ESB has the ability to deploy a rule-based agent as a distributed rule inference service, and install it as a Web-based endpoint on the Mule object broker; hence it can support the communication in this rule-based agent processing network via its transport protocols. Mule ESB allows converting messages into a format that is usable by the services/agents and provides a highly scalable and flexible application messaging framework. This can be used to communicate synchronously or asynchronously among the local agents and agents/services on the Web [20].

Several agents run a rule engine at their core, and these agents are installed as Mule components which are active at configured Mule endpoints, e.g. JMS endpoints. Mule ESB provides a large variety of transport protocols, including JMS, SOAP, HTTP, which can be used to transport the messages (queries, answers) to the registered endpoints or external applications. In our EnviroPlanner, JMS is used for the internal agent communication, while HTTP is used for external data sources.
Chapter 3

Rule Languages and Tools

This chapter describes the Semantic Web rule languages and tools that are relevant to this thesis. Section 3.1 provides a brief introduction to rules, followed by rule languages in Section 3.2. This is followed by a discussion of Semantic Web tools that are used in this thesis for rule inferencing in Section 3.3.

3.1 Rules

Rules and ontologies are two main schemes for representing different kinds of knowledge. Rules are appropriate for representing logical implications, or associating actions with conditions under which the actions should be taken [58]. Rules are also playing an important role in the Semantic Web. Rule-based markup languages and inference systems allow information to be shared and published on the Web. Rule languages relevant to this thesis are described in this chapter.
3.2 Rule Languages

Rule languages enable users to constrain, share, modify, and map data in various Semantic formats. Rule languages used by our system are discussed below.

3.2.1 Rule Markup Language

Rule Markup Language (RuleML), defined by the Rule Markup Initiative\(^1\), is an XML-based markup language for rule-based and knowledge-based systems. It allows users to store, interchange, and retrieve information on the Web \([14]\).

RuleML is implemented with XML Schema, and Relax NG \([12]\), a simple schema language for XML which is based on RELAX and TREX; it permits users to realize high-precision Web rule interchange. Horn logic’s Datalog sublanguage, a language in the intersection of SQL and Prolog, is at the semantic foundation of RuleML. RuleML uses Datalog as part of its family of sublanguages. In Datalog, we can define facts, corresponding to rows of relational tables and rules, corresponding to tables defined implicitly by views \([20]\). The entire relation constitutes an atomic formula, marked by angle brackets \(<Atom>....</Atom>\), where participating tags define the following categories:

- **Atom**: Atomic formula
- **Rel**: Relation constant
- **Ind**: Individual constant
- **Data**: Data constant

The following RuleML rule implies that a person buys an product from a merchant if the merchant sells the product to the person \([28]\).

\(^{1}\)http://www.ruleml.org
One of the main advantages of RuleML is that it is a platform independent markup language and can bridge between other standards, e.g. between W3C RIF and ISO Common Logic, which can be achieved by the XSL Transformations (XSLT) [32, 24]. Reaction rules introduce actions into the RuleML family [26]. In Enviro-Planner, an agent communicates with other agents through sending and receiving Reaction RuleML messages.
3.2.2 Reaction RuleML

Reaction RuleML works as a medium of communication between users and agents. It is an extended XML-serialized sub-language of RuleML for the family of reaction rules that supports actions, reactions, action logic rules, and events [55]. The building blocks of Reaction RuleML are [54]:

- One general (reaction) rule form which is Rule. Specialized forms can be created for production rules, messaging rules, etc. One general reaction rule can be specialized to e.g. production rules, trigger rules, ECA rules, and messaging rules.

- Three execution styles defined by the attribute @style
  - Active: Actively polls/detects occurred events in global Event, Condition, and Action (ECA) style
  - Messaging: Waits for incoming complex event messages and sends out-bound messages as actions
  - Reasoning: Knowledge representation derivation and event/action logic reasoning and transitions

- Messages <Message> defines inbound or outbound event message

A Reaction rule might apply globally or locally nested within other rules. The general format of a Reaction rule consists of six partially optional parts:

```xml
<Rule style="active" evaluation="strong">
  <label> <!-- metadata --></label>
  <scope> <!-- scope --></scope>
```

27
Inbound and outbound messages (<Message>) are used to interchange events and rulebases (modules) between the agent nodes [56]; these messages have the following format:

<Message mode ="outbound" directive ="pragmatic performative">
  <oid> <!-- conversation ID --> </oid>
  <protocol> <!-- transport protocol --> </protocol>
  <sender> <!-- sender agent/service --> </sender>
  <content> <!-- message payload --> </content>
</Message>
3.2.3 Positional-Slotted Language (POSL)

POSL integrates Prolog’s positional and F-logic’s slotted methods of representing knowledge (facts and rules) on the Semantic Web. It is based on the mathematical notions of positional relations and logical inference enriched by object-centered frames similar to the slotted syntax of F-logic [9]. The main advantage of POSL is its conciseness for knowledge ranging from the relational-logical to object-centered modeling styles. As a human-readable presentation syntax, POSL is faster to write and easier to read than any XML syntax [27]. Yet, RuleML/XML can be generated from it URL.

3.3 Semantic Web Tools and Rule Engines

Semantic Web rule languages describe human-readable knowledge and machine-formalized knowledge. Using the rule engines, these rule languages can be used to interchange and share knowledge through a distributed network.

3.3.1 Prova

Prova [48] is a Java Virtual Machine (JVM) based rule language for reactive agents and event processing [1], but Prova works as both a (Semantic) Web rule language and a highly expressive distributed (Semantic) Web rule engine. Prolog is used to facilitate both rule translation processes and rule component identifications [46], which enable Prova to support distributed inference services, rule interchange, complex reaction rule-based workflows, rule-based complex event pro-
cessing, dynamic access to external data sources, Web Services, rule-based decision logic, and Java APIs.

Prova combines declarative rules, ontologies, and inferences with dynamic object-oriented programming like Java, and follows the design principle of the Semantic Web initiatives [44]. Prova can be run in a plain Java environment as a standalone application, as an Open Services Gateway initiative (OSGI) component, or as a rule inference service on the Rule Responder ESB. According to [31], the Prova rule engine supports the following rule types:

- 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
- 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 conversation-based workflow reaction and behavioral logic, based on complex event processing

### 3.3.1.1 General Prova Features

The Prova language has four main components [47]:

**Simple (atomic) terms:**

Prova includes two types of simple terms: constants and variables. Inter-
nally, they are Java interface Prova-Constant and Prova-Variable respectively.

**Compound terms (lists):**

Prova keeps lists in arrays instead of recursive lists. Lists are represented in a standard Prolog way as `[Head—Rest]`. *Head* is one or more (comma-separated) elements that are the list's starting point, and the *Rest* is the list tail.

**Facts:**

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

\[ \text{predicate}(\text{arg}_1, \text{arg}_2, \text{arg}_3, \ldots \text{arg}_n) \]  

(3.1)

**Rules:** They are written in the standard Prolog like way. In the following example `[-]` is pronounced as IF.

\[ \text{headLiteral}(\text{args}_h) : - \text{bodyLiteral1}(\text{args}_1), \]
\[ \text{bodyLiteral1}(\text{args}_2), \]
\[ \ldots, \]
\[ \text{bodyLiteraln}(\text{args}_n) \]  

(3.2)

Here, if a rule has no body, then it is a *fact*, and if the rule has no head, it is a *goal*.  

31
For better understanding, the following Prova code snippet is coded in an Organizational Agent (OA) to match a incoming query with its existing interfaces (facts). When OA receives the query, it tries to match the query with one of its existing interfaces; if no match is found, it sends a message to the user and the rule fails.

```prova
processMessage(XID,From,Primitive,[XIArgs]) :-
  not(interface([XIArgs], ModeDeclarations, Description)),
  sendMsg(XID, esb, From, "answer",
    noPublicInterface(interface ([XIArgs]))),
  sendMsg(XID, esb, From, "no_further_answers", [XIArgs])),
  fail().
```

The `processMessage(XID,From,Primitive,[XIArgs])` rule process the user’s query. When the `processMessage(....)` rule is initialized, it executes its body to check if the facts are true; if true, it ends with the `fail()` to prevent any further processing of the query. Here, the interface describes the correct format for the user query which is accepted by the QA. The parameters of the `processMessage()` rule are described below:

- **XID**: Name of the OA (e.g. The FrederictonRegion or SaintJohnRegion of EnviroPlanner)
- **From**: Name of the endpoint
- **Primitive**: User name
- **X**: Relation name surrounded by an angle bracket as in `<Rel>.....</Rel>`
• Args: Arguments of the relation <Ind>, <Var>, <Expr>, etc.

If no match is found among the existing interfaces, the system informs the user via the EA that his/her request has failed because it does not have an interface. At the same time, a second sendMsg(....) is executed with the performative no further answers; thus, EnviroPlanner will stop searching for the public interface pattern instead of waiting for more answers.

3.3.1.2 Reactive messaging

Reactive messaging is a fundamental part of the system that is used to organize distributed Prova engines into a network of communicating agents. It also provides the foundation for the rule-based workflow and event processing functionality [47]. A Prova agent is an instance of a running rulebase, where agents communicate with other agents through a protocol; basically an argument is translated into message passing primitives, both for receiving and sending. In our system, an ESB is used for communication; therefore, the distribution aspect of a Prova agent is not part of the Prova-agent rulebase code of our system. Inside a Prova-agent rulebase, the key principle used for message processing is pattern matching. Users specify parameters (XID, protocol, Destination, Performative, and Payload) as constants, and leave the ones that require further reasoning as free variables. Sending messages reverses this process, where users include the same above parameters [47]. Sending and receiving messages is discussed below.
3.3.1.3 Sending Messages

Message sending is initiated when a Prova engine processes the literal `sendMsg`, which is a premise for any rule. The duality with message receiving is visible in sharing the same structure of mandatory position-based arguments. The arguments of the built-in `sendMsg(....)` are discussed below.

Conversation ID

Conversation ID is used for request and response message correlation. In our system, this is a constant value; therefore, the sent message is a follow-up to an ongoing conversation so that a particular conversation ID can be obtained from an earlier received message.

Protocol

Currently, Prova supports three internal protocols: task, async, and swing. The swing protocol cannot be presently used for message sending. It is only used for receiving notifications from swing components. Our system uses only two protocols which are discussed below:

- Async Protocol: It is used to pin the processing to a unique thread as calculated from the conversation ID. This protocol is preferred to handle ongoing conversations, maximize data locality, achieve correctness, and minimize the synchronization.

- The ESB Protocol: It is used for designating the agent as the forwarder for dispatching the message.
In the Rule Responder environment, Mule ESB components are used as a container for Prova agent; thus, with the protocol set to esb, and destination set to the logical end point of the Mule ESB, the message is delivered by Mule to the endpoint of an agent. When a Mule component receives a message from elsewhere, the inbound protocol name is swapped from esb to async. It guarantees that the conversation is processed using the async protocol. The Prova agent then understands that, although it receives messages on a nominally internal async protocol, it needs to use the esb protocol for responding.

**Destination**

For internal intra-agent messaging, the destination must be either a constant zero or a running agent [47]. In our system, the destinations are the agent names. For instance, interchanging messages between the OA and PA requires reasoning with some rules so that the appropriate PA can be found to answer the query. To do so, the OA utilizes an assigned(....) predicate, which is informally described as a Prolog rule below (its actual definition is in Java).

\[
\text{assigned(XID,Responsibility,Role,Agent,Result):­} \\
\qquad \text{import(URL),} \\
\qquad \text{reasoner(Reasoner),} \\
\qquad \text{rdf(URL,Reasoner,Responsibility,Role,Agent,Result).} \\
\]

When the assigned(....) predicate is triggered, the sub-rule import(....) retrieves the address of the ontology, which includes roles and responsibilities of the PAs for a particular Sub-OA or Super-OA. After getting the URL of the ontology, the
sub-rule $reasoner(Reasoner)$ retrieves the rule language in which the OWL ontology is implemented. In our system, we used the light Web Ontology Language (OWL). Then the built-in $rdf(....)$ retrieves the name and role of the responsible agent based on the responsibility assigned to it in the RAM.

When Prova gets the responsible PA name, the destination would be the logical name of that PA’s endpoint. Since Mule ESB is used for message routing, the logical name of that PA’s endpoint is configured via the Mule configuration file. A PA is configured inside the Mule configuration as a topic.

**Performative**

Performatives are message types that agents can interpret and act upon by actually inspecting the payload of a message [47]. Prova allows agent developers to encode the performative-based reasoning. Processing messages identified as performative usually involve two steps: the receiving process and the sending process. In our system, the FrederictonRegion Sub-OA, MonctonRegion Sub-OA, and the SaintJohnRegion Sub-OA receive answers from their PA, and send them back to the EA through the Super-OA.

**Payload**

Payload is the main information content of a message. In Prova, the content of a query message must be a ProvaList, which works as a container to hold other data. Each element in a ProvaList container has to be either a Prova constant, variable or another list. The following example provides a better illustration:
rcvMsg(XID, esb, From, Performative, [X|Args]) :-
  understandPerformative(XID, From, Performative, [X|Args]),
  rcvMsg(XID, esb, Agent, no_further_answers, Payload),
  sendMsg(XID, esb, From, no_further_answers, [X|Args]),

In the list [X|Args], the X and the Args can be any term (e.g. list, var, or constant). These terms can be repeated as many times as the number of terms in the user query [20]. In our system, this payload is the matched Prova list for the users RuleML-based query.

3.3.1.4 Receiving Messages

Prova uses reaction rules for both sending and receiving messages. Prova reaction rules are like Horn rules but they use the head literal, with the predicate “rcvMsg”, and a number of body literals. EnviroPlanner uses two types of reaction rules: global and inline reaction rules.

Global Reaction Rules

The simplest form of a reaction rule is the one where the head of the rule is a message receiving primitive, distinguished by the predicate symbol rcvMsg. When rcvMsg(...) is executed, the function understandPerformative(...) interprets the Performative of the message to determine the process of the payload [X|Args]. To accept messages from a sender, the rcvMsg() rule has five main parameters: XID, Protocol, From, Performative, and Payload. These five parameters have been described at the beginning of this chapter. The rcvMsg rule has a rulebase lifetime scope so that it can be active as long as the rulebase runs in a Prova engine. The
rule is ready to receive any number of messages as they arrive at the agent which indicates its global scope [22].

**Inline Reaction Rules**

Inline reaction rules are useful for workflow and event processing. They are more dynamic and volatile; therefore, their scope can be controlled in a variety of ways, including restricting them to accept just one message, restricting them to a specified number of messages, or limiting them by a timeout [47].

Inline reaction rules are comparable to closures or continuations. The reaction is created as part of evaluating the body of a rule when a \texttt{rcvMsg} is a part of that body. EnviroPlanner uses the message \texttt{rcvMsg} as a rule's premise that itself has the head “rcvMsg” to limit the lifetime scope of receiving answers. This \texttt{rcvMsg} executes twice; the first time is for any parameters' value, and the second time is for the parameters that match the \texttt{rcvMsg} constants. If constants are matched, the next following premises are executed; if the parameters do not match, then the rule is terminated by the system.

Throughout the execution process of the rule, the OA understands that the agent has no more answers when the OA receives a message in the \texttt{rcvMsg} premise that has performative equal to \texttt{no.further.answers}; the OA concludes that no more answers would be sent by that agent. The first premises \texttt{rcvMsg(....)} wait till the the last premise \texttt{rcvMsg(....)} is matched which 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 inside the \texttt{rcvMsg(....)} [20]. To make the correlation among messages, it uses the \textit{XID} as a conversation ID. Hence, including \texttt{rcvMsg(....)} in the body of any Prova rule results in the creation of a temporal reaction that freezes the current state of all the context and body literals following \texttt{rcvMsg} [22].

Temporal reaction is just a stored data. It is matched against a Prova agent when it perceives a new inbound message on the matching protocol. Once the matching message is received, that stored data gets fully erased right after it is used. To keep the reaction rule invoked indefinitely, \texttt{rcvMult} rule is used because \texttt{rcvMsg} works only once.

In EnviroPlanner, OAs use reaction rules for sending and receiving queries to/from the External Agent and appropriate Personal Agents (PAs). OAs use RAM to find the responsible PA who is able to answer the query. Next, OAs send the query to the PA, obtain the query answer from the PA, and deliver it back to the EA.

### 3.3.2 OO jDREW

OO jDREW [37] is an Object Oriented extension of jDREW (Java Deductive Reasoning Engine for the Web). It thus works as a Java-based reasoning engine for executing RuleML facts and rules including their object-centered slotted extensions [38]. The Personal Agents (PAs) of EnviroPlanner are implemented using the OO jDREW reasoning engine. OO jDREW supports both bottom-up and top-down
reasoning.

- **Bottom-Up**: This execution method is used to infer all derivable facts from a set of clauses (forward reasoning).

- **Top-Down**: This is used to solve a query over the knowledge base by reducing it to subqueries down to the level of facts (backward reasoning).

Both execution modes are useful depending on the user application requirements. In our system, we used the top-down approach due to the query focus of Rule Responder. Before processing a query, the OO jDREW engine parses the RuleML query message and translates it into an OO jDREW structure. Then, OO jDREW loads the PA’s profile and, based on facts/rules defined in the profiles, derives the answer.

Each RuleML query message contains a performative wrapper that is used to interpret the message. The message has the following information [20]:

- **Sender**: Sender of the message

- **Protocol**: Transport protocol used by the ESB

- **OID**: Conversation ID of the message

- **Content**: Incoming query payload of a message or outgoing answer payload of a message

- **Mode**: Outbound or inbound direction of a message

- **Directive**: Distinguishes whether a message content is a query or an answer (further directives would be needed in an extended Rule Responder)
3.3.3 XSLT

Extensible Stylesheet Language Transformations (XSLT) [19] is a XML-based language which is used for transforming XML documents into other XML documents. It represents the W3C recommendations for defining XML document transformation and presentation. XSLT consists of three parts:

- XSL Transformations (XSLT): A language for transforming XML.
- The XML Path Language (XPath): It is a simple expression language used by XSLT to access or to identify parts of a XML document. XPath cannot be used stand-alone. It is always used in the context of a language like XSLT.
- XSL Formatting Objects (XSL-FO): It is a XML vocabulary for specifying formatting semantics.

XSLT is designed to be usable independently of XSL-FO. It can also be used as part of the XSL. When both XSLT and XSL are used together, XSL specifies the styling or formatting vocabulary of a XML document. The XSL uses XSLT to describe how the document is transformed into another XML document that uses the formatting XML vocabulary XSL-FO [50]. Figure 3.1 shows a simple transformation of a XML document into another XML document.

A transformation expressed in XSLT is called a stylesheet which describes rules for transforming a source tree into a result tree. A XML transformation is achieved by associating patterns with templates. Stylesheets contain a set of template rules and each template rule has two parts: a pattern part which is matched against
nodes in the source tree, and a template part which is instantiated to form a part of the result tree. This pattern matching against a template allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures [19].

EnviroPlanner uses XSLT to transform XML documents (RuleML) into Prova and vice versa. In particular, the user query format of EnviroPlanner is in RuleML/XML; hence, when a user poses a query to the Super-OA through the EA, the Super-OA cannot process the query without transforming it into Prova because the Super-OA processes only Prova rules. To solve this, XSLT is used to allow the interaction between the EA and the Super-OA. Analogously, XSLT is used for transforming messages among other agents of EnviroPlanner.

![Figure 3.1: Process Flow of XSLT](image-url)
Chapter 4

EnviroPlanner Architecture

This chapter describes the overall architecture of our system along with its agents and queries. Figure 4.1 displays the top-level architectural view of EnviroPlanner that includes a system of Personal Agents (PAs) and Organizational Agents (OAs); these agents are accessed through an External Agent (EA). Within the Organizational Agents, one is distinguished as the Super-Organizational Agent (Super-OA). EnviroPlanner employs the Super-OA on top of Sub-OAs; these Sub-OAs deal with incoming queries for the Fredericton, Moncton, and Saint John regions; each region has its local rulebases. The Super-OA, which acts as a dispatcher, filters and redirects incoming queries to the Sub-OAs.

The Super-OA returns the query answer to the EA, hence to the user, when it obtains the answer from the Sub-OAs. The Sub-OA for each region delegates queries to Personal Agents (PAs), one each for air-, soil-, water-, and energy-related queries pertaining to a region. The RAM is used to describe the roles and responsibilities
Figure 4.1: EnviroPlanner Architecture
of the PAs in EnviroPlanner. Each PA acts as a semi-autonomous agent that has its own rule-based decision logic in an (instantiation of a) rule engine to execute the environmental facts and rules of the agent.

EnviroPlanner is built on top of an Enterprise Service Bus (ESB) communication middleware known as Mule [7] which can seamlessly handle message-based interactions between agents. Mule includes improvements in customizable schemas for connectors, simpler invocation of connectors, a new polling mechanism, a powerful DataMapper for translating payloads from one data format to another format, a message filter, message enrichment, route capabilities, and a powerful logging facility. Mule ESB transports Reaction RuleML messages from one endpoint to another endpoint of agents or external data sources. Mule also provides strong security protocols for HTTP, HTTPS, and JMS, during message transmission.

4.1 Knowledge Representation

In our system, knowledge is represented in KB format which consists of rules and ontologies. EnviroPlanner employs Prova as its OA language, and is executed on the rule engine with the same name. The OAs use a Responsibility Assignment Matrix (RAM) to deliver the incoming environmental query to the appropriate PA. To do so, the Prova rule engine executes the air, soil, water and energy experts' KB profiles, and retrieves the information from the responsible PA so that user queries can be answered. This is achieved through the use of a Web Ontology Language (OWL Lite) KB that is used to bind the respective roles and responsibilities of an
agent by typed variables in the agent’s rule logic.

4.2 Distributed System

A distributed system consists of a collection of various component, connected through a network and distribution middleware, which enables nodes to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility [2]. Figure 4.2 shows a simple distributed system where each component is connected through a common middleware.

![Figure 4.2: Architecture of a Simple Distributed System [41]](image)

Common characteristics of a distributed system are as follows [2]:

46
- Resource Sharing: Hardware, software, or data can be used anywhere in the system
- Openness: System accepts extensions and improvements
- Concurrency: Components access and update shared resources
- Scalability: Components do not need to be changed when scale of a system increases
- Fault Tolerance: System is available even at low levels of hardware, software, or network reliability
- Transparency: System should be perceived by users and application programmers as a whole rather than as a collection of cooperating components

Based on these common characteristics, distributed system architectures can take the form either star or mesh topology architecture [36]. These architectures are briefly described below.

### 4.2.1 Distributed System Architecture

Distributed system architecture has rapidly evolved by supporting, combining, and applying heterogeneous architectures. The combination of distributed systems and Semantic Web techniques can be viewed as a way of extending the capabilities of distributed systems for representing rule-based content communicated among distributed systems connected through a network. A brief discussion of various distributed system architecture approaches given below.
Client-Server Architecture
It connects all nodes/clients with a central hub. The main advantage of the client-server architecture is that one malfunctioning node/client does not affect the rest of the network. After getting a request from the client, the server does the required processing to gather the requested information, and sends it back to the client.

In our system, a star-topology-like architecture connects the air, soil, water, and energy PAs to the Sub-OAs. Figure 4.3 shows how PAs are connected to the Sub-OAs in EnviroPlanner.

![Figure 4.3: Star Topology](image)

Networked-Mesh Architecture
A networked architecture can use full mesh topology, partial mesh topology, or star topology (similar to Figure 4.3). In a full mesh topology, each node is connected with every other nodes, which creates redundant connections. If any link fails, information can flow through many other links to reach its destination. However, one of its main disadvantages is that it is a very complex architecture, especially when a very large number of nodes are connected with each other. Because of its complex architecture, it is not a pragmatic approach to exchange information. To
overcome this issue, partial mesh topology is more efficient.

4.2.2 Distributed Rule System

To increase the efficiency of rule-based inference systems, their KBs have been distributed, which pushed forward the application of rule technologies to distributed systems and multi-agent systems [23]. EnviroPlanner is designed and implemented as a distributed rule-based system, and its loosely coupled inference engines execute the rule bases to retrieve required information.

Unlike in centralized systems where data or knowledge bases are stored in a single location, the rule bases of the experts' PAs can be stored at distributed locations. Advantages of a distributed system [36] include the ease of maintenance as well as the ability to achieve fault-tolerance and improved efficiency through distributed processing. Distributed maintenance allows agents to update their facts and rules without affecting the rulebases of their agents. If all facts and rules were stored in one centralized rule base, problems experienced by a single agent would affect the entire system.

After receiving a user query, if an OA finds that the PA responsible for answering the query is not available, an extended Rule Responder system could try to contact another available PA that might be able to answer the user query. If the OA would not receive an answer within a predefined time frame, it can request a timeout and start looking for another PA that could answer the query. At the end, if the OA would not find any PA to answer the query, the system could tell the user that
no answer was found. This process would affect the overall performance of the system as it encounters frequent timeouts to look for an appropriate PA. However, the repeated PA-search overhead would not be noticeable to the External Agent.

In our system, each Organizational Agent can employ its own vocabularies defined as a Semantic Web ontology to give its facts/rules an environmental domain-specific meaning. These vocabularies can be used for the conversation with other agents to enable a semantic and pragmatic interpretation of the messages. Rulebases’ distribution approach provides a robust solution that ensures a more fault tolerant system. By using distributed rule bases, we are able to ensure the system is less prone to problems and with multiple rule engines translating knowledge bases instead of one speed up the query processing. For the deployment of all distributed agents on the Web and for communication in agent networks, EnviroPlanner uses the Mule Enterprise Service Bus (ESB) middleware, which transports rulebases, queries, and answers among agents.

4.3 Organizational Agents

Organizational Agents (OAs) represent EnviroPlanner as a Virtual Environmental Organization (VEO). For example, an OA contains a Knowledge Base (KB) that describes the policies, and opportunities. An OA manages its local Personal Agents (PAs), provides control of their life cycle, and ensures overall goals and policies of the organization and its semiotic structures [31]. OAs, which require high levels of expressiveness to represent the logic of semi-automated agents, are
implemented using the Prova rule engine.

An example query that the OA can answer for the EnviroPlanner is, “What are the hydro power stations in Saint John region?”. When the Super-OA of EnviroPlanner receives this user query through an EA in Reaction RuleML format, the Super-OA checks the Prova knowledge base and the RAM to determine which Sub-OA is responsible for answering queries about the Saint John region. Queries about Rothesay, Hampton, Grand Bay, and St. George are also mapped into the Saint John region where the relationship is many to one. These cities are into one region, and that is why the Super-OA mapped them into one Sub-OA. When the Super-OA determines the responsible Sub-OA for answering the query, it delivers that query to the Sub-OA. The selection logic for the delegation of users queries to PAs is described by the RAM, and the OA selects a responsible agent based on the RAM. After receiving the above query from the Super-OA, the Saint John region Sub-OA first determines who is the responsible expert for answering energy-related queries. Topics coming into the energy PA can be ‘hydro’, ‘nuclear’, ‘fuel oil’, and ‘Coal’. These topics are mapped to one role which is energy. Because these topics are related to energy, they are mapped to the same address of the energy expert’s PA. When the appropriate expert for energy-related queries is found, the Saint John region Sub-OA delegates the query to the Personal Agent of the energy expert. Once the query is processed, the PA responds with the list of all hydro power stations in the Saint John region.

OAs act as a dispatchers, where user queries can propagate from the EA, to the
Super-OA, to one of the Sub-OAs, and to the managed sets of local PAs. We use Prova messaging reaction rules to manage the communication flow in the OAs, and these reaction rules work as a coordination pattern for user queries delegation to OAs through the EA. OAs utilize the RAM to delegate the user query to the responsible PA.

EnviroPlanner uses two types of OA: the Super Organizational Agent (Super-OA), and Sub Organizational Agents (Sub-OA). Both are now described in detail.

4.3.1 Super-Organizational Agent

To delegate user queries to the appropriate Sub-OA, and to avoid the duplication of rules after abstracting ('lifting') generalized environmental knowledge from the Sub-OAs, EnviroPlanner employs an Organizational Agent that is distinguished as the Super-Organizational Agent (Super-OA), on top of its hierarchical architecture. The Super-OA manages the Fredericton, Moncton, and Saint John regions' Sub-OAs, which work underneath the Super-OA. Upon receiving a query from the External Agent, the Super-OA makes the decision to which Sub-OA the query should be delivered. The Super-OA utilizes its Sub-OA selection criteria based on the incoming query properties. To analyze and deliver the incoming query to the suitable Sub-OA, the Super-OA uses the processMessage Prova rule, which is given below:

\[
\text{processMessage}(\text{XID, From, Primitive, [Function|Arguments]}) : - \\
\text{concat([ep_, Function], FunctionwithNamespace),} \\
\text{assigned(XID, FunctionwithNamespace},
\]

52
"ep_responsible", Agent, Result),
CandidateSubOAs=ws.prova.mule.impl.DecisionCriteria.
subOADecision(Result, Function, Arguments),
element(CandidateSubOA, CandidateSubOAs),
sendMsg(XID, esb, CandidateSubOA, "query",
   [Function|Arguments]),
rcvMult(XID, esb, CandidateSubOA, "answer", Answer),
sendMsg(XID, esb, From, "answer", Answer).

In the above example, taken from our EnviroPlanner, the processMessage premise calls its CandidateSubOAs premise, which performs the suitable Sub-OA selection. This selection eventually leads the incoming query to be delegated to the suitable PA. To make the Sub-OA selection decision, the CandidateSubOAs premise checks the available query properties. When the Super-OA cannot make the decision through the CandidateSubOAs, it consults with its Responsibility Assignment Matrix to make the Sub-OAs selection decision.

4.3.2 Sub-Organizational Agent

EnviroPlanner consists of an extension and the SymposiumPlanner-2011/2012-like instantiation of the Rule Responder framework, which is similar with regards to the agents’ communication architecture but not with regards to the knowledge domain. The SymposiumPlanner system [52] uses two Sub-OAs for question-answering. We extended the framework that allows developer to specify an arbitrary number of Sub-OAs. This provides architectural flexibility to developers so that they can add
as many Sub-OAs as needed for answering user queries about different locations or regions. Figure 4.4 shows the generalized way of connecting Sub-OAs with the Super-OA in EnviroPlanner.

In our system, we use three Sub-Organizational Agents (Sub-OAs); one for Fredericton, one for Moncton, and another one for Saint John region environmental topics. The Super-OA redirects incoming query to either one of these Sub-OAs. Using the Prova rule engine execution environment, the Fredericton, Moncton, and Saint John region Sub-OAs constitute an intelligent filtering and dispatching system so that they can receive incoming messages, directly answer some queries or selectively delegate them to the most suitable PA. Furthermore, each Sub-OA has its own Prova rulebase to process incoming queries.

All three Sub-OAs work under the Super-OA, and map incoming environmental queries, which are related to the air, soil, water, and energy of their respective region, to the PA responsible for that query. Each Sub-OA manages the air-, soil-, water-, and energy-focused PAs of its region, sends queries to, and receives answers
from, a selected PA after a solution is obtained by the PA from its local KB. The Sub-OAs are aware of the local knowledge-based capabilities of the underlying PAs, and send back the query results to the EA through the Super-OA after obtaining the results from the appropriate PA.

### 4.3.3 Responsibility Assignment Matrix

To define various roles and responsibilities of the agents, EnviroPlanner implements a Responsibility Assignment Matrix (RAM) that supports the OAs to map an incoming query to the appropriate PA whose local KB is most suitable to answer a user query. A standard RAM is a matrix that answers the question, “which agent will be responsible for answering a user query?” EnviroPlanner uses Web Ontology Language Lite (OWL Lite) to represent the responsibilities of the agents, thus defining the RAM, and OAs can access it through the Semantic Web built-ins of Prova. Key properties defined in a RAM are described below [31]:

- **Responsible**: This property is for the agent who does the work to achieve the assigned task. EnviroPlanner assigns exactly one agent as responsible for a task, yet other agents could be delegated to assist in the task required. For example, an energy expert can be responsible for answering queries containing keywords for hydro, wind, fossil, or nuclear power but (s)he could be supportive to queries related to air pollutants. In our system, each PA is owned by a human expert (or expert subteam). The responsible owner(s) use(s) a PA as a proxy for getting queries answered based on pre-encoded expert knowledge.
• **Accountable**: This property points to the agent that is actually accountable for the correct and thorough completion of a deliverable or task. In our system, this role is played by the OA, which receives the answer from the PAs and further processes it before forwarding it to the EA.

• **Informed**: This property points to the agent that is kept up-to-date on progress on completion of the task, just using one-way communication. For this, EnviroPlanner assigns the EA, which is informed about the result by the Super-OA.

Responsibilities are bound by the knowledge representation of the Prova to a set of typed variables in the agents’ rule logic. In the RAM/OWL specifications of EnviroPlanner, specialized tasks are assigned to their respective agents.

Using the special Prova RDF query built-in or the Prova SPARQL query built-in, an agent’s responsibility in EnviroPlanner can be returned. To coordinate the responsibilities of the agents with incoming queries, EnviroPlanner has a RAM/OWL ontology specifications for each OA, as detailed in the following.

**Organizational Agent Responsibility Assignment Matrix**

The main class of the OWL ontology is the *EnviroPlannerSystem*, which is the root of all sub-classes.

```owl
<owl:Class rdf:ID="EnviroPlannerSystem">
```

The above mentioned *EnviroPlannerSystem* class has a *FunctionManagement* subclass that defines its subordinate subclasses to assign tasks to the agents. The
FunctionManagement subclass acts as the root subclass of EnviroPlanner.

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

For instance, EnviroPlanner uses the energyServices task/relation to retrieve the energy services of the regions. Therefore, before retrieving this information, energyServices task/relation is first declared as a subclass of FunctionManagement.

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

In the RAM, the responsibility of a PA is mapped on the Super-OA level in the following way:

<owl:ObjectProperty rdf:ID="responsible">
  <rdf:type rdf:resource="owl#FunctionalProperty">
  <rdfs:domain rdf:resource="#FunctionManagement">
  <rdfs:range rdf:resource="#epRegionATcityB;EP-REGION-SAINTJOHN"/>
</owl:ObjectProperty>

The hydroPowerPlants task is mapped to the EP-REGION-SAINTJOHN Sub-OA, which is responsible for delivering the Saint John-related queries to its subordinate PAs, and the task is delivered to the suitable PA that is able to answer the energy-related query. Analogously, the rest of the tasks is mapped to responsible Sub-OAs, which deliver them to the appropriate PAs.
A RAM’s ‘responsible’ properties, represented in an OWL Lite ontology, are accessed by a Super-OA or Sub-OA via querying with the Prova ‘assigned’ built-in predicate, mapping query topics to, respectively, the responsible Sub-OA or PA.

### 4.3.4 Performatives

In chapter 3.3.1.3, we mentioned that performatives describe the envelope of a message content. As a prototype modeling a distributed multi-agent VEO, OAs and PA of EnviroPlanner are execute by different rule engines, which help them to share and connect with different local data sources. These agents communicate with each other globally based on the common Reaction RuleML interchange language which carries the performatives from one endpoint to another endpoint of an agent by the Mule ESB. Receiving agents use these performatives to understand the context of incoming messages. The main language constructs of messaging Reaction Rules are:

- `sendMsg()`: Send outbound message
- `rcvMsg()`: Receive an inbound message
- `rcvMult`: Receive one or more inbound messages

Processing messages with performatives typically involve a sending and a receiving process.
sendMsg(XID, Protocol, Agent, Performative, Payload|Context)
rcvMsg(XID, Protocol, From, Performative, Payload|Context)
rcvMult(XID, Protocol, From, Performative, Payload|Context)

In the above code snippet, \textit{XID} is the conversation ID, \textit{Protocol} is the communication protocol, and \textit{Performative} is the pragmatic envelope for the message content. A standard nomenclature of performatives is FIPA Agents Communication Language (ACL). \textit{Payload} is the message content sent in the message envelope.

4.4 Personal Agents

In the EnviroPlanner system, various profiles of PAs are built for the Fredericton, Moncton, and Saint John regions. Air-, soil-, water-, and energy-related PAs work as the assistants of experts on each of these topics. A useful sample of the environmental facts and rules is included in each PA’s profile. These environmental fact- and rule-based profiles are interpreted by the experts’ PAs to answer routine queries in his or her stead. Such queries are delegated to the most knowledgeable PA by the Sub-OA of EnviroPlanner using the RAM.

To represent simple facts, e.g., an expert’s contact details and relevant personal information, we use Friend of a Friend (FOAF)-like [4] profiles. These profiles are used to represent relevant information about a person of the EnviroPlanner VEO as facts, with rules being layered on the respective profiles. The FOAF-extending profiles have access to the RDF and OWL specifications.
Every PA of EnviroPlanner contains a knowledge base for supporting the PA-owning human expert by applying the (expert-encoded) knowledge. Each PA uses its KB and rule engine to deduce query results. Besides the KB, the rule engine can access different data sources and derive answers based on the local rule-based decision logic of the respective PA.

4.4.1 Translation Between EnviroPlanner Agents

Agents can use different local rule engines and rule languages. In order to communicate globally with other agents, however, agents need to use a common rule interchange language. In our system, RuleML is used as the common rule interchange language; thus, environmental rulebases, queries, and answers need to be translated from and to RuleML so that they can be interchanged with other agents.

In EnviroPlanner, Reaction RuleML provides the translator service framework with a Web form interface accepting predefined selection-based rule templates for the communication with the EA, as well as Servlet HTTP interfaces. These Servlet HTTP interfaces can be used for translation into and from platform-specific rule languages such as Prova [53]. User queries to EnviroPlanner can be formulated via templates which are translated into a Discourse Representation Structure (DRS) [63]. The DRS gives a logical/structural representation of the controlled English. 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 [20].

To establish a communication between agents, the translator services use different translation technologies such as XSLT stylesheets to translate from and to Reaction RuleML. The general syntax of a reaction rule consists of the following six partially optional parts [53]:

```xml
<Rule style ="activelmessaging|reasoning"
     eval ="strong|weak|defeasible|fuzzy">
  <oid> <!-- object id --> </oid>
  <label> <!meta data of the rule --></label>
  <scope> <!- scope of the rule --></scope>
  <qualification> <!-priorities, validity--> </qualification>
  <quantification> <!-variable bindings--> </quantification>
  <on> <!-event part --> </on>
  <if> <!- condition part --> </if>
  <then> <!- logical condition --> </then>
  <do> <!- action part --> </do>
  <after> <!- postcondition part --> </after>
</Rule>
```

The general syntax of the reaction rule code snippet are described below:

- The Rule attribute is a general (reaction) rule form
• Reaction rule can have three execution styles defined by the attribute \textit{style};
  By default it is reasoning.

• The \textit{weak}, \textit{strong}, \textit{defeasible}, and \textit{fuzzy} attributes are the evaluation semantics

• Oid, label, scope, and qualification are optional attributes

• Messages define inbound or outbound event message

Different types of reaction rules can be obtained from this general rule syntax. EnviroPlanner, like other Rule Responder instantiations, uses the general message syntax of Reaction RuleML for distributed rule-based agents communication. Inbound and outbound messages are used to interchange information and rulebases between the agent nodes. The message syntax, used by EnviroPlanner, is shown below:

\begin{verbatim}
<Message>
  <oid><!-- Conversation id --></oid>
  <protocol><!-- Used Protocol --></protocol>
  <agent><!-- sender/receiver agent/service --></agent>
  <directive><!-- pragmatic primitive --></directive>
  <content><!-- Message payload --></content>
</Message>
\end{verbatim}

Using these messages, agents can interchange information as well as complete rulebases. Agents can engage in long-running asynchronous conversations and keep track of them by using the following XML attributes (indicated by “@” signs):
• Oid, etc.: Conversation ID to manage conversation state

• Protocol: Defines the message passing and coordination protocol of a conversation

• Directive: Corresponds to the pragmatic instruction. In other words, the directive attribute characterizes the meaning of the message

To illustrate the asynchronous conversations between agents, an environmental query to EnviroPlanner is given below, based on the above message syntax.

```xml
<Message mode="outbound" directive="query-sync">
  <oid><Ind>EnviroPlanner</Ind></oid>
  <protocol><Ind>esb</Ind></protocol>
  <sender><Ind>User</Ind></sender>
  <content>
    <Atom>
      <Rel>powerPlants</Rel>
      <Var>PowerPlantName</Var>
      <Var>Capacity</Var>
      <Var>PowerPlantTypes</Var>
    </Atom>
  </content>
</Message>
```

63
The Reaction RuleML translator services are configured in the transport channels of the inbound and outbound links of the deployed rule engines on the Enterprise Service Bus. To configure the Reaction RuleML translator services, EnviroPlanner employs the Mule Enterprise Service Bus that is described in the Enterprise Service Bus section of this chapter. Incoming Reaction RuleML messages (receive) are translated into platform-specific rulebases which are executed by the Prova rule engine, and outgoing rulebases (send) are translated into Reaction RuleML in the outbound channels before they are transferred via a selected transport protocol such as HTTP or JMS (among many others).

4.4.2 Query Answering for Personal Agents

The knowledge base of facts and rules of each profile under a Personal Agent is local to that profile, which is written in Positional Slotted Language (POSL). Air-, soil-, water-, and energy-related PAs can use two methods to process the query and send the answers to the respective Sub-OA. One of them computes all of the solutions and sends all of the answers back to that Sub-OA, one answer at a time. After sending the last answer, the PA sends an end-of-transmission message to the Sub-OA, which indicates that no more answer messages would be sent by the
PA. EnviroPlanner does not use this method because computing all answers before sending any of them might cause an answer-delivery timeout, especially if there is an infinite enumeration of the solutions. To avoid this issue, EnviroPlanner implements interleaved backtracking with transmission. After receiving a query from the Sub-OA, the respective PA starts computing the first solution. When the first solution is found, the PA immediately sends the solution message to the Sub-OA. The PA starts computing for the next solutions while the earlier one is being transferred. Analogously, when all solutions are found in a finite interleaved enumeration, the PA sends an end-of-transmission message. After receiving as many answers as needed from an infinite enumeration, the Sub-OA can send a no-more-answer message to the PA, which stops the computation of further solutions of the PA. Otherwise, the Sub-OA will receive as many answers as possible before a PA timeout or similar exception will occur.

The PA sends back a failure message to the Sub-OA when it receives a query but cannot compute any answers for that query. In those exceptional cases, the Sub-OA could try to send the same query to another PA, etc. If no answer is found, the Sub-OA sends a failure message to the EA through the Super-OA. In our system, the Fredericton, Moncton, and Saint John region Sub-OAs receive the answers from the PAs and deliver them to the Super-OA that eventually delivers the answers to the EA.
4.5 External Agent

The EA is the point-of-entry that allows a user to pose a query to the Organizational Agents of EnviroPlanner. The EA is based on a Web interface that allows users to pose environmental queries. On its Web interface, it employs a menu-based Web form, which uses JavaScript to generate both an controlled English description of the environmental query and a RuleML/XML query but users can modify the the RuleML/XML query. Figure 6.1 shows the Web interface (EA) of EnviroPlanner. After translation to an equivalent RuleML/XML message, this message is sent to the OA via HTTP interface’s Post or Get method. In addition, the EA allows users to pose region-specific queries directly to one of the appropriate Sub-OAs; by selecting the query from the Sub-Organizational Agent (@FrederictonRegion), or Sub-Organizational Agent (@MonctonRegion), or Sub-Organizational Agent (@SaintJohnRegion), respectively.

When a user clicks on the “submit” button to pose a query to EnviroPlanner, the EA communicates with the OA and sends the query to the OA. After the query is processed by various agents of EnviroPlanner, the OA sends the answer back to the EA. Therefore, the EA facilitates the process of querying expert profiles about air, soil, water, and energy topics. Upon receiving the answers, the EA shows the final results to the user.
EnviroPlanner Demo

EnviroPlanner is an instantiation of the emerging Rule Planner, an extended Semantic Web framework for distributed rules and knowledge bases structured by ontologies for environmental querying. Citizens thus are enabled to acquire the EnviroPlanner’s agents and experts about environmental issues such as air, energy, water, and so on.

EnviroPlanner utilizes the Mule\(^1\) Java-based open-source Enterprise Service Bus (ESB) as its communication middleware. The Mule ESB allows developers to integrate different agents/services/applications by putting a communication bus between them. This allows system to communicate without having dependencies

\(^1\)http://www.mulesoft.org/

Figure 4.5: External Agent of EnviroPlanner

4.6 Enterprise Service Bus

EnviroPlanner utilizes the Mule\(^1\) Java-based open-source Enterprise Service Bus (ESB) as its communication middleware. The Mule ESB allows developers to integrate different agents/services/applications by putting a communication bus between them. This allows system to communicate without having dependencies

\(^1\)http://www.mulesoft.org/
on agents or knowledge of other disparate systems on the bus. Figure 4.6 shows the overview of a Mule ESB.

![Mule ESB Diagram]

Figure 4.6: Mule Enterprise Service Bus [40]

Mule is a messaging platform-based ESB architecture, but it goes beyond the typical ESB functionalities as a transit system for carrying data between agents/services/applications. Mule ESB has powerful capabilities that include [7]:

- Service creation and hosting: As a lightweight service container, it can display and host reusable services.

- Service mediation: This is a security service for message formats and protocols that enables location-independent service calls.

- Message routing: It routes, filters, aggregates, and re-sequences messages based on the content and rules.
• Data transformation: It exchanges data across various formats and transport protocols.

It provides a distributable object broker to manage all sorts of service components such as the agent services of EnviroPlanner. Figure 4.7 illustrates the transport mechanism of the Mule ESB that enables it to process incoming messages to the proper outgoing application/agent.

![Figure 4.7: Transport Mechanism of Mule [51]](image)

Figure 4.7 also shows the logic order of processing, from receiving an incoming message to sending the message to the right endpoint. Mule has three processing modes, which are described below:

• **Asynchronous**: In this mode, many events (messages) can be processed by
the same component at the same time in various threads.

- **Synchronous**: When a message object component receives an event message, the whole request is executed in a single thread.

- **Request-Response**: This mode allows a message object component to make a specific request for an event and wait for a specified time to get a response.

The object broker follows the Staged Event Driven Architecture (SEDA) pattern [61]. SEDA decomposes a complex, event driven application into a set of stages connected by queues and supports massive concurrency demands on Web-based services, as well as providing a highly scalable approach for asynchronous communication. Figure 4.8 shows a simplified breakdown of the integration of Mule into the EnviroPlanner architecture.

Distributed agents are deployed as Mule components which are listened at configured endpoints, e.g., JMS endpoints, HTTP ports, JDBC database interfaces, etc. Since EnviroPlanner uses the Reaction RuleML as its common rule interchange language format between the agents, translator services are used to translate inbound and outbound messages from Reaction RuleML into platform specific execution syntaxes of rule engines and vice versa. In our case, translator services are based on Extensible Stylesheet Language Translation (XSLT). The Mule architecture consists of the following core components [1]:

- **Component**: Contains the business logic, e.g., a bean, a service or a Plain Old Java Object (POJO), a Java object which does not follow any of the
major Java object models, conventions, or frameworks [8].

- **Transport**: Handles the connectivity protocols such as FTP, email, HTTP, SOAP, JDBC, JMS/MQ, and CORBA.

- **Inbound Router**: Controls received messages before they are sent to the service.

- **Transformers**: Transforms inbound messages into formats required by processing components, or transforms results of the processing components into the formats of the outbound messages.

- **Filters**: Filters can be used to express the rules for filtering messages on routers.

- **Outbound Router**: Determines where a message needs to be sent after it is

---

**Figure 4.8: Mule Integration in EnviroPlanner [70]**

---

71
processed by the service; may include various kinds of filters to test whether
the message can be sent to a given endpoint.

- A transport endpoint receives a message.

When a message is received by the transport endpoint, it is transformed to the
required format and forwarded to the inbound router where the message is pro­
cessed. Next, the inbound router forwards this message to the component where
filters can be used to express the rules for filtering messages or applies its rule
logic to the message and sends to the outbound router. Then the outbound router
delegates the message to the target endpoint.

To have a better understanding of how endpoints are configured in our system,
the Mule configuration of EnviroPlanner is elaborated below:

```xml
<global-property name="http.host" value="localhost"/>
<global-property name="MULE_PORT" value="8888"/>
<global-property name="MULEIMPLEMENTATION"
    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 the above code snippet, the global environmental properties, such as the ad­
dress of our Tomcat installation, the port number assigned to the Mule ESB are
configured in our mule-all-config file. In order to create endpoints for PAs, Mule
requires these properties to be characterized as topics. For example, air and its
global properties are a topic. In EnviroPlanner, each PA is responsible for a topic
about which the PA is the appropriate one to answer an environmental query. For instance, we have four topics names for air, soil, water, and energy-related PA respectively for Fredericton region:

<global-property name="FREDERICTONREGIONEANAME"
value="FREDERICTON-REGION"/>
<global-property name="FREDERICTONREGIONOANAME"
value="FREDERICTONREGION"/>
<global-property name="FREDERICTONREGION_PORT" value="9678"/>
<global-property name="FREDERICTONREGION_Prova"
value="rules/use_caseEPREGIONFREDERICTON/
EP-REGION-FREDERICTON_Organization/
EP-REGION-FREDERICTON-Responder.prova"/>
<global-property name="FREDERICTONREGIONTopicName"
value="FREDERICTONREGION"/>
<global-property name="FREDERICTONREGIONTopic1" value="Air"/>
<global-property name="FREDERICTONREGIONTopic2" value="Energy"/>
<global-property name="FREDERICTONREGIONTopic3" value="Soil"/>
<global-property name="FREDERICTONREGIONTopic4" value="Water"/>

EnviroPlanner uses Java Message Service (JMS) as the main transport protocol.

<jms:activemq-connector name="jmsConnector"
specification="1.1" brokerURL="${jms.url}"/>
Since the Mule ESB is used for message routing, the logical name of the endpoint as configured via the Mule configuration file is the destination; topic names are the destination (agent) in the following code snippet. The Mule configuration also contains service endpoints.

<jms:endpoint name="EP-REGION-FREDERICTON"
    topic="${FREDERICTONREGIONTopicName}"/>
<http:endpoint name="epREGIONATfredericton_Air"
    address="${TOMCAT}/${FREDERICTONREGIONOAName}$
    {FREDERICTONREGIONTopic1}"/>
<http:endpoint name="epREGIONATfredericton_Energy"
    address="${TOMCAT}/${FREDERICTONREGIONOAName}$
    {FREDERICTONREGIONTopic2}"/>
<http:endpoint name="epREGIONATfredericton_Soil"
    address="${TOMCAT}/${FREDERICTONREGIONOAName}$
    {FREDERICTONREGIONTopic3}"/>
<http:endpoint name="epREGIONATfredericton_Water"
    address="${TOMCAT}/${FREDERICTONREGIONOAName}$
    {FREDERICTONREGIONTopic4}"/>

The endpoint descriptors are used to create a model for the Mule configuration to allow communication between the EA and that specific model for its underlying topics (e.g., air, soil, water, and energy PAs).

<model name="FREDERICTON-REGIONModel">
Mule provides a large variety of transport protocols that can be used to transport the messages to the destination endpoints or external applications. EnviroPlanner uses JMS for the internal communication between distributed agent instances. The usual processing style is asynchronous using SEDA event queues. However, sometimes synchronous communication is needed. For instance, to handle communication with external synchronous HTTP clients such as Web browsers where requests, e.g., by a Web from, are sent through a synchronous channel. In this case, a synchronous bridge component dispatches the requests into the asynchronous messaging framework and collects all answers from the internal service nodes, while
keeping the synchronous channel with the external service open. After all asyn-
chronous answers have been collected, they are sent back to the still connected
external service via the HTTP-synchronous channel.
Chapter 5

Local and Global Knowledge Bases

5.1 Fact- and Rule-Based Scripts for PAs

In our system, locally distributed KBs for PAs are stored in the experts’ PA profiles. Each expert’s profile is called local knowledge base because facts/rules inside the profile are specific for the respective expert and these facts/rules are not globally shared, unlike the Prova KB. Each PA acts in a rule-governed manner on behalf of the expert represented in the form of a profile written in the Positional Slotted Language (POSL). EnviroPlanner uses the Super-OA to handle incoming queries and outgoing answers to and from Sub-OAs with their underlying PAs. EnviroPlanner allows the PAs to automatically respond to requests using the query-related facts and rules of the experts’ profiles. In this section, these profiles are explained with illustrative examples.
To represent simple facts, like a VEO’s chairman details and relevant personal information, we use FOAF-like profiles, a vocabulary that can be used to describe people, organizations, and their relationships to each other. These profiles can be used to get relevant details about a person from the VEO as facts, while rules can be layered on top of these profiles. In the following example, EnviroPlanner uses the PA’s FOAF-like profile to get the relevant details of the public affairs’ chairperson of Environment New Brunswick (ENB).

For example, when a user poses an inquiry about the chairperson of ENB, the respective PA invokes this rule and related facts from the KB profile of the expert. Using these facts, EnviroPlanner finds out the head of the organization and his/her personal details. In the following rule, the first argument (input) of the 3-ary person predicate is the branch name of the organization where the person works, the second argument (input) is the name of the organization, and the third argument (output) is the personal details of that person.

\[
\text{person(?BranchName, ?OrgName, ?PersonalDetails) :-}
\]
\[
\text{partOf(?BranchName, ?OrgName),}
\]
\[
\text{chair(?BranchName, ?PersonalDetails).}
\]

FOAF-like 2-ary facts about the public affair’s chairperson of ENB, Fredericton

\[
\text{partOf(PublicAffairsENBF, FrederictonRegion).}
\]
\[
\text{chair(PublicAffairsENBF,}
\]
\[
\text{PersonalDetails[foafffirstName[Sarah],}
\]

78
In the above code snippet, we are displaying the following information.

- Fredericton region is the part of the public affairs division of Environment New Brunswick, Fredericton (ENBF)
- Sarah Ali is the chair of the public affairs division of ENBF
- Her title is \textit{dr}
- Her email address is \textit{Sarah.Ali@tenb.ca}
- Her telephone number is \textit{15061231111}

Based on the above FOAF-like facts, if a person poses a query to know the contact details of the head of public affairs division of ENB, Fredericton region, EnviroPlanner shows her personal and contact details.

Now we shall discuss another kind of rule that is part of the air PA’s profile. The air PA uses the backtracking mechanism for finding multiple solutions from the knowledge base. In order to solve a query, the PA first attempts to find the first clause of a knowledge base, it being the first subgoal of rules. When all necessary variables are matched with their respective values, the PA checks the second subgoal of the clause through the last subgoal. If the PA failed to find a match for the respective variable, the clause fails. As a result, the PA backtracks.
To have a better understanding of how the backtracking mechanism works in our local knowledge base, we illustrated the `emissionNeutralizer` knowledge base below.

```prolog
emit("AB Chemical Limited":string, 05:real, PhosphoricAcid).
emit("AB Chemical Limited":string, 08:real, HydrobromicAcid).
emit("AB Chemical Limited":string, 10:real, NitricAcid).
emit("AB Chemical Limited":string, 14:real, CarbonDioxide).
emit("AB Chemical Limited":string, 18:real, SulfuricAcid).
emit("NB Pulp and Paper Limited":string, 10:real, SulfuricAcid).
emit("NB Pulp and Paper Limited":string, 02:real, HydrobromicAcid).
chemReact(NitricAcid, SodiumHydroxide, SodiumNitrate, ton).
chemReact(NitricAcid, PotassiumHydroxide, PotassiumNitrate, ton).
chemReact(CarbonDioxide, PotassiumHydroxide, PotassiumBicarbonate, ton).
chemReact(CarbonDioxide, LithiumHydroxide, LithiumCarbonate, ton).
chemReact(SulfuricAcid, SodiumHydroxide, SodiumSulphate, ton).
chemReact(CarbonicAcid, MagnesiumHydroxide, MagnesiumCarbonate, ton).
chemReact(Hydrochloric, Chlorine, HypochlorousAcid, ton).
chemReact(CarbonicAcid, HydrogenCarbonates, Bicarbonate, ton).
```

In the following code snippet, first we get the amount of pollutants from a factory that emits pollutants and then we test for a remedy for a corresponding (equally named) pollutant. We have two premises sharing a join variable; they match in a way where the output of the first one is the input of the second one because there is
a factory associated with a pollutant, and there is a pollutant again associated with a remedy. Therefore, at some point, emissionNeutralizer finds matching remedies but sometimes it does not find a remedy in the second group of facts.

multiply(?AmountofRemedy:real, ?AmountofPollutant:real, 0.20:real).

In the first premise, a factory emits a certain amount of pollutants,

A remedy is necessary to neutralize the pollutants emitted by the factory. In the following premise, chemical reaction between the pollutant and remedy results in another chemical compound.


In the third premise, the ‘multiply’ built-in is used to calculate the amount of remedy the neutralization process needs to neutralize the amount of pollutant. The multiply built-in takes the AmountofPollutant and it is twenty percent as input and generates the amount of remedies which will be needed for neutralization process as output.
multiply(?AmountofRemedy:real, ?AmountofPollutant:real, 0.20:real)

In order to find a remedy for a pollutant, emissionNeutralizer first attempts to find the predicates and arguments. When it gets the predicates, it binds the arguments of the predicates with the corresponding variables. If it does not find the corresponding variable for an argument, it goes back to its last choice point and checks if there is an alternative variables to find a remedy for the given pollutant.

In order to solve the above, emissionNeutralizer first attempts to find an emit fact. This binds Factory to AB Chemical Limited, AmountofPollutant to 05, and Pollutant to PhosphoricAcid. emissionNeutralizer then attempts the goalchemReact and tries to bind Pollutant to PhosphoricAcid, remedy to SodiumHydroxide, ChemCompound to SodiumNitrate, and UnitofMeasurement to Ton. This will fail, since it does not find a remedy for PhosphoricAcid in the above knowledge base.

As a result, it backtracks. This means that it goes back to its last choice point and sees if there is an alternative solution. In this case, this means going back and attempting to find another emit fact. This time, emissionNeutralizer uses the second emit fact, binding Factory to AB Chemical Limited, AmountofPollutant to 08, and Pollutant to HydrobromicAcid. This causes emissionNeutralizer to try the goal chemReact(). Again this will fail to match our knowledge base. As a result, emissionNeutralizer backtracks again. This time, it finds the third emit fact, and bind Factory to AB Chemical Limited, AmountofPollutant to 10, Pollutant to NitricAcid, and attempt the goal chemReact(). This goal matches against third chem-React fact. As a result, emissionNeutralizer will succeed with Factory=AB Chemical Limited, Pollutant=NitricAcid, Remedy=SodiumHydroxide,

In the above facts, we have Hydrochloric and Carbonic acid as remedies which can be neutralized by Chlorine and Hydrogen Carbonates, respectively; but these remedies are not used by emissionNeutralizer because no factories use Hydrochloric acid, and/or Carbonic acid.

EnviroPlanner can draw conclusions from rule-sets when all conditions for a query are satisfied; rule sets allow us to make conditional statements about a domain. EnviroPlanner uses inference rules to generate air pollutant emission information. The air expert’s KB profile contains the airPollutantEmission rule that is related to the amount of total air pollutant emission and greenhouse gas emission.


add(?TotalGHGEmission:integer, ?AmountofCO2:integer,

The airPollutantEmission rule displays the air pollutant emission information of a given year based on a given company name; below we briefly discuss how the rule works to show the output.

The source premise checks if the company name and the year, in this case, ABC Corporation and 2012, respectively, are in the air expert's profile of the Fredericton Sub-OA. To show the air pollutant emission information of ABC Corporation in 2012, the source premise checks for ABC Corporation as a company name and 2012 as a year in the air expert's KB profile; moreover, it checks the facility name of the company and how much air pollutant is released in air by the facility.

source("ABC Corporation":string, facilityName["ABC Petroleum Limited":string], 2012:integer, ton, 90:integer).
gHGEmission(facilityName["ABC Petroleum Limited":string],

gHGEmission(facilityName["ABC Pulp and Paper Mill":string],
When a company has multiple facilities, as in this case, ABC Petroleum Limited and ABC Pulp and Paper Mill, the source premise checks for the results of all facilities under the same company name in the same year. Based on the facility names of the company and the year, the greenhouse gas emission premise, $gHGE\text{-mission}$, checks the amount of the main greenhouse gases released by the facilities in a given year. In the third premise, the 'add' built-in is used to calculate the amount of total greenhouse gas emission by adding up the amount of Carbon dioxide, Methane, Nitrogen dioxide, and other greenhouse gases. The forth premise verifies that total emission is greater than total greenhouse gases emission. Once all the arguments are found in the air expert's KB profile under $airPollutantE\text{-mission}$, EnviroPlanner displays the results of the query to the user; here, $ton$ was used as a unit of measure to display the amount of the released air pollutants.

For illustrative purposes, we display the query about air pollutant emission below along with responses of the system.

```
<Message mode="outbound"
directive="query-sync">
<oid><Ind>EnviroPlanner</Ind></oid>
<protocol><Ind>esb</Ind></protocol>
<sender><Ind>User</Ind></sender>
<content>
<Atom>
  <Rel>airPollutantEmission</Rel>
  <Ind type="integer">2012</Ind>
</Atom>
```
The above query on interaction with the system receives the following responses where the first response shows the amount of air pollutant emission caused by one of the industrial facilities of ABC Corporation, ABC Petroleum Limited.

```
<Message mode="outbound" directive="answer">
  <oid><Ind>httpEndpoint:4</Ind></oid>
  <protocol><Ind>esb</Ind></protocol>
  <sender><Ind>httpEndpoint</Ind></sender>
  <content>
    86
  </content>
</Message>
```
<Atom>
  <Rel>airPollutantEmission</Rel>
  <Ind type="integer">2012</Ind>
  <Ind type="string">ABC Corporation</Ind>
  <Expr>
    <Fun>facilityName</Fun>
    <Ind type="string">ABC Petroleum Limited</Ind>
  </Expr>
  <Ind>ton</Ind>
  <Expr>
    <Fun>greenHouseGas</Fun>
    <Ind type="integer">90</Ind>
  </Expr>
</Atom>

The following response is the second response, which shows the values of the air pollutant emission caused by another industrial facility of ABC Corporation, ABC Pulp and Paper Mill.
<Message mode="outbound" directive="answer">
  <oid><Ind>httpEndpoint:4</Ind></oid>
  <protocol><Ind>esb</Ind></protocol>
  <sender><Ind>httpEndpoint</Ind></sender>
  <content>
    <Atom>
      <Rel>airPollutantEmission</Rel>
      <Ind type="integer">2012</Ind>
      <Ind type="string">ABC Corporation</Ind>
      <Expr>
        <Fun>facilityName</Fun>
        <Ind type="string">ABC Pulp and Paper Mill</Ind>
      </Expr>
      <Ind>ton</Ind>
      <Expr>
        <Fun>greenHouseGas</Fun>
        <Ind type="integer">30</Ind>
        <Ind type="integer">6</Ind>
        <Ind type="integer">5</Ind>
        <Ind type="integer">7</Ind>
        <Ind type="integer">12</Ind>
      </Expr>
      <Ind type="integer">70</Ind>
    </Atom>
  </content>
</Message>
5.1.1 Fact and Rule Processing in a Profile

We showed, in the previous section, how facts and rules are stored in the experts’ profiles as locally distributed KBs. In this section we discuss how these facts and rules are processed in the Prova KB of the Sub-OAs. To have a better understanding of the functionality of the Sub-OAs, we present the structure of a Prova KB in a Sub-OA.

Prova allows us to import rulebases, which are managed as modules at the beginning of the Prova knowledge base. The following rule bases are imported in our Prova knowledge base.

:-eval(consult('..../..../ContractLog/math.prova')).
:-eval(consult('..../..../ContractLog/update.prova')).
:-eval(consult('..../..../ContractLog/utils.prova')).
:-eval(consult('..../..../ContractLog/owl.prova')).

In the above code snippet, Prova is importing multiple Prova rulebases for decision making at the knowledge base level. Using performatives, the Prova KB also describes the pragmatic envelope for the message content of a query.

In our system, air, soil, water, and energy PAs run local rule engines which can access different sources of local data and compute answers according to the local
rule-based decision flow of the PAs. EnviroPlanner uses OO jDREW as its local rule engine. However, arbitrary rule engines can be used by the PA if they have query interfaces to ask queries and receive answers. In order to communicate with other agents, these queries and answers need to be translated into common Reaction RuleML interchange format. Reaction RuleML provides an interface definition language for system-wide agent communication. In the interface definition language, modes are states of instantiation of the predicate described by mode declarations knowledge base. The public interface knowledge base contains ‘+’(s) and ‘-’(s) which shows the use of constants and variables respectively; the “+” term is intended to be input, the “-” term is intended to be output, and the “?” term is undefined/arbitrary which means it can be either input or output.

The External Agent accesses EnviroPlanner through the public interfaces, which reveal abstracted information to users and hide local information from users. An interface defined in the Sub-Organizational Agent for the air pollutant emission is given below.

```plaintext
interface(airPollutantEmission(Year, CompanyName, FacilityName,
    UnitofMeasure, greenHouseGas(TotalGHGEmission,
    AmountofCO2, AmountofCH4, AmountofN20,
    AmountofOtherGHGases), TotalEmission),
    airPollutantEmission("+","+","-","-",
    greenHouseGas("-","-","-","-", "-"), "-"),
    "request air pollutant emission released by
    the company in the given year").
```
The above interface is designed as a response to the query which is issued by the EA. The interface matches the relation with the topic and the request for retrieving information for the topic in question. When the topic is mapped with the RAM, the query is sent by the RAM to the Sub-OA which contains the interface to handle this request.

In the \textit{airPollutantEmission} query, the argument 2012 and \textit{ABC Corporation} are intended for input “+” so that it can be mapped through the RAM to the appropriate Sub-OA. The variables \textit{FacilityName}, \textit{UnitofMeasure}, \textit{AmountofCO2}, \textit{AmountofCH4}, \textit{AmountofN2O}, \textit{AmountofOtherGHGases}, and \textit{TotalEmission} are intended for specifying output; hence the use of “-”.

\subsection{Illustrative Example}

For illustration purposes, we discuss the \textit{airPollutantEmission} rule from the Fredericton region air expert local KB.

\begin{verbatim}
                   ?TotalEmission) :-
  source(?CompanyName:string, ?FacilityName, ?Year:integer,
           ?UnitofMeasure, ?TotalEmission:integer),
  gHGEmission(?FacilityName, ?Year:integer, ?AmountofCO2,
                ?AmountofCH4, ?AmountofN2O,

\end{verbatim}
?AmountofOtherGHGases:integer),
add(?TotalGHGEmission:integer, ?AmountofCO2:integer,
?AmountofCH4:integer, ?AmountofN20:integer,
?AmountofOtherGHGases:integer),

The 'airPollutantEmission' query is received by the EA, and the EA delivers the query to the Super-OA. The selection criteria of the Sub-OAs are specified in the Prova rule base, and the RAM which is described in the OWL Lite, on the Super-OA level. Based on the RAM, EnviroPlanner processes the query to decide which Sub-OA is suitable to deliver the query. A fact is used to point Prova to the OWL ontology file used by the Fredericton Sub-OA. The OWL ontology file represents a RAM which is used by the Sub-OA to map an incoming query to the appropriate PA. We explain the Prova-RAM interaction in the next section.

On the Sub-OA level of EnviroPlanner, the 'airPollutantEmission' query is handled by a Prova rule base which is given below:

processMessage(XID, From, Primitive, airPollutantEmission(Year,
CompanyName, FacilityName, UnitofMeasure,
greenHouseGas(TotalGHGEmission, AmountofCO2,
AmountofCH4, AmountofN20,
AmountofOtherGHGases), TotalEmission)) :-
!, println(["----------------------------------------------------------"]),
println([" QUERY RECEIVED          "]),
println(["----------------------------------------------------------"]),
println(["Received message from browser."]),
println("Looking up Responsible Personal Agent"),
assigned(XID,Agent,epREGIONATfredericton_AirPollutant,
    epREGIONATfredericton_responsible),
println("Responsible Personal Agent Found: ",Agent),

sendMsg(XID,esb,Agent, "query", airPollutantEmission(Year,
    CompanyName, FacilityName, UnitofMeasure,
    greenHouseGas(TotalGHGEmission, AmountofCO2,
    AmountofCH4, AmountofN20,
    AmountofOtherGHGases), TotalEmission)),
println("Sent message to the right ",Agent," PA.")},
println(" "),

rcvMult(XID,esb,Agent,"answer", substitutions(FacilityName,
    UnitofMeasure,TotoalGHGEmission,AmountofCO2,AmountofCH4,
    AmountofN20, AmountofOtherGHGases, TotalEmission)),
println("Received message from the ",Agent," PA.")},

sendMsg(XID,esb,From, "answer",airPollutantEmission(Year,
    CompanyName, FacilityName, UnitofMeasure,
    greenHouseGas(TotoalGHGEmission, AmountofCO2,
    AmountofCH4, AmountofN20,
    AmountofOtherGHGases), TotalEmission)),

println(" ")
println(["Sent message to the browser."])．

The `processMessage()` function, presented in the above code snippet, has the following parameters:

- **XID** - Conversation ID for the correlation, which keeps track of the ongoing conversation
- **From** - Sender of the message which can be an agent or a service endpoint
- **Primitive** - Pragmatic instruction of the message
- **airPollutantEmission** - This is the relation of the query, which describes the pragmatic envelope for the message content
- **assigned()** - Find the responsible agent
- **sendMsg()** - Message sending is initiated when the Prova engine processes this literal
- **rcvMult** - Receive one or more than one context-dependent multiple inbound messages.

In the `assigned()` part, the rule looks at the conversation id, the agent, in this case Fredericton region air expert’s PA, that is responsible for answering the question based on its role in the RAM. In response the `sendMsg()` premise asks the agent that was discovered above to answer the `airPollutantEmission` query. If there is more than one answer, the `rcvMult()` premise allows the PA to respond with multiple answers in response to the above question. Each time the `rcvMult()` premise receives an answer from the PA, `sendMsg()` sends the answer to the EA,
until no more answers are left. The \texttt{println()} premise prints the status of message handling process.

\section*{5.2 OA Global Knowledge Base}

In EnviroPlanner, global knowledge bases contain knowledge/facts relevant to environmental experts. Our system uses both ontologies and rules on the OA level, which are globally shared via the OAs to assist the PAs to process user queries. A subset of rules is distributed amongst the PAs of the human experts knowledge base profiles. Shared ontologies and rules on the OA level are referred as the global KBs. Global knowledge in our system works as a combination of ontologies and rules. Ontologies capture the knowledge domain of EnviroPlanner, and rule arguments are defined by statements and signatures.

In our system, we used Prova to define the rules in the global knowledge base. Prova provides support for Java class hierarchies and ontological type systems (e.g. OWL or RDFS ontologies). The OWL2Prova API supports integration of Semantic Web ontologies written in RDFS or OWL. It has a rich built-in library for query languages such as SQL, SPARQL, and XQuery [30], and it is designed to work with the distributed Enterprise Service Bus and OSGi environments [47]. Each Prova knowledge base file is pointed by the the Mule configuration file. In our case, the configuration file is \texttt{mule-all-config.xml}.

In the following code snippet, we present a few global property names and their respective values that we used during the Mule configuration. The Prova KB file
for Fredericton region is pointed in the *mule-all-config.xml* file.

```xml
<global-property name= "FREDERICTONREGIONEAName"
 value= "FREDERICTON-REGION"/>
<global-property name= "FREDERICTONREGIONOAName"
 value= "FREDERICTONREGION"/>
<global-property name= "FREDERICTONREGION_PORT"
 value= "9678"/>
<global-property name= "EPREGIONFREDERICTON_Prova"
 value= "rules/use_caseEPREGIONFREDERICTON/
 EP-REGION-FREDERICTON/
 EP-REGION-FREDERICTON-Responder.prova"/>
<global-property name= "FREDERICTONREGIONTopicName"
 value= "FREDERICTONREGION"/>
<global-property name= "FREDERICTONREGIONTopic1" value= "Air"/>  
<global-property name= "FREDERICTONREGIONTopic2" value= "Soil"/>  
<global-property name= "FREDERICTONREGIONTopic3" value= "Water"/>  
<global-property name= "FREDERICTONREGIONTopic4" value= "Energy"/>
```

### 5.2.1 Prova-RAM Interaction

A fact is used to point Prova to the OWL ontology file used by the particular Sub-OA. The OWL ontology file represents a RAM which is used by the Sub-OA to map an incoming query to the appropriate PA. The assignments of the ontology for the Sub-OAs in the global knowledge bases carried out in the following way:
   EP-REGION-FREDERICTON.owl").
   EP-REGION-SAINTJOHN.owl").
   EP-REGION-MONCTON.owl").

5.2.2 Prova Public Interface

Prova uses a public interface to access PA rules. The interface defines the format of an expected query which can be issued to EnviroPlanner. It has the following format:

interface(performative(Performative),
   performative("\?"), "description")

To further elucidate the arrangement, let us look at an interface for contact information of an expert.

The first argument is the performative which checks the variables of the user queries.

interface(airPollutantEmission(Year, CompanyName, FacilityName,
   UnitofMeasure, greenHouseGas(
   TotalGHGEmission, AmountofCO2,
   AmountofCH4, AmountofN20,
   AmountofOtherGHGases),
   TotalEmission)
The second argument describes the relation with +'s and -'s to represent constants and variables.

\[
\text{airPollutantEmission}(\text{"+","+","-","-"}, \text{greenHouseGas}(\text{"-","-","-","-"}, \\
\text{"-","-","-"}, \text{"-"}), \\
\text{"-"}), \\
\]

The last argument is the description of the framework.

request air pollutant emission released by the company in the given year

As a whole, the Prova public interface looks like the following code snippet.

\[
\text{interface}(\text{airPollutantEmission}(\text{Year, CompanyName, FacilityName,} \\
\text{UnitofMeasure, greenHouseGas(} \\
\text{TotalGHGEmission, AmountofCO2,} \\
\text{AmountofCH4, AmountofN20,} \\
\text{AmountofOtherGHGases),} \\
\text{TotalEmission}), \\
\text{airPollutantEmission}(\text{"+","+","-","-"}, \text{greenHouseGas}(\text{"-","-","-","-"}, \\
\text{"-","-","-"}, \text{"-"}), \\
\text{"-"}), \\
\text{"request air pollutant emission released by the company in the given year"}). \\
\]

5.2.3 Query Processing in Prova Knowledge Base

An incoming query is handled by the \text{processMessage()} rule. The structure of the \text{processMessage()} rule is as follows:

\[
\text{processMessage(XID,From,UserName,performative(Performative)):-} \\
\]
• XID - OA name

• From - Endpoint name

• Username - Primitive (e.g. User)

• performative - The relation name enclosed within $Rel$

• Performative - Contents of relations (Ind, Val, Expr, Plex, etc.)

On receiving a query, the system consults the RAM to delegate tasks to PAs.

assigned(XID,Agent,Responsibility,Role),

where,

• XID - OA name

• Agent - Name of found agent

• Responsibility - Responsibility name for query

• Role - The role (agent) 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 predicate: Reaction rules which react to inbound messages

• rcvMult predicate: In-line reactions in the body of messaging reaction rules to receive one or more multiple inbound event messages
A message is then sent to the selected/found agent:

\[
\text{sendMsg}(\text{XID}, \text{esb}, \text{Agent}, \text{Type}, \text{performative(Performative)}),
\]

Here,

- \text{XID} - OA name
- \text{esb} - Transport protocol
- \text{Agent} - Name of found agent
- \text{Type} - Query
- \text{performative} - Relation name of query
- \text{Performative} - Content of relation

The answer is then received from the Agent:

\[
\text{rcvMult}(\text{XID}, \text{esb}, \text{Agent}, \text{Type}, \text{performative(Performative)}),
\]

Here,

- \text{XID} - OA name
- \text{esb} - Transport protocol
- \text{Agent} - Name of found agent
- \text{Type} - Answer
- \text{performative} - Relation name of query
- \text{Performative} - Content of relation
This answer is then routed back to the EA:

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

Here,

- **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 content of the relation
Chapter 6

User Interface and
System Evaluation

6.1 EnviroPlanner User Interface

For deploying agents on the Web and enabling communication among them, EnviroPlanner uses the Mule ESB and utilizes messaging from Reaction RuleML for communication between the distributed agent services. For human interaction, our system utilizes dynamic HTML forms as the Web user interface which is shown in Figure 6.1. Users are able to select interfaces and fill in the query parameters for interaction with the system. Queries are translated into Reaction RuleML based on the Rule Responder interface description file and values entered by users.
Figure 6.1: EnviroPlanner User Interface
6.1.1 Query Description in Controlled English

EnviroPlanner’s Web interface allows users to compose queries by employing a menu-based form, which uses JavaScript to generate RuleML format and controlled English descriptions of the queries. To have a better understanding of a controlled English description for a query, we provided both the RuleML format of the query, and the controlled English description for the same query. For example, when the user selects the *air pollutant emission* query from the menu-based Web form, EnviroPlanner generates the following RuleML query format, and a controlled English description of the query.

```xml
<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:ruleml2011="http://ibis.in.tum.de/projects/paw#">
  <Message mode="outbound" directive="query-sync">
    <oid><Ind>EnviroPlanner</Ind></oid>
    <protocol><Ind>esb</Ind></protocol>
    <sender><Ind>User</Ind></sender>
    <content>
      <Atom>
        <Rel>airPollutantEmission</Rel>
        <Ind type="integer">2012</Ind>
        <Ind type="string">ABC Corporation</Ind>
      </Atom>
    </content>
  </Message>
</RuleML>
```

104
For the above RuleML-formated query, the controlled English description is as follows:

*How much air pollutants were released in 2012 by ABC Corporation?*

*Show me the total amount of greenhouse gases, and individual amounts of the following major greenhouse gases released by the same company in the same year:*

*Amount of Carbon monoxide*

*Amount of Methane*

*Amount of Nitrogen dioxide*

*Amount of other greenhouse gases*

*Total emission caused by the ABC Corporation*
6.2 System Evaluation

In this section, we present some experimental results of EnviroPlanner. Our illustrations of the operations are based on knowledge bases. EnviroPlanner is built on top of Rule Responder which has a good overall performance. However, during the evaluation process of our system, different user queries show various levels of performance. While evaluating the system, it was observed that different user queries can lead to various amounts of time to display the results. Users can use the interface, which utilizes its Java-script to change the query based on user choice, to pose a query to the system. When users change a query in the menu-based Web form, the change is instantaneous, and a brief description of the query is displayed on the interface in controlled English.

In our system, the PAs interact with local knowledge bases. The OAs' performance depends on the required time for interchanging messages with PAs and the EA. The OAs translate messages between endpoints using the Prova rule engine, and this process can vary depending on the user queries. Though EnviroPlanner uses Reaction RuleML for system wide interaction, the OAs and PAs use different rule engines to process messages. There is a predefined time frame for the OAs; therefore, when the OAs do not get answers from PAs within the timeout interval, Mule closes the current communication channel between the OA and the EA, which eventually leads to the termination of user queries.
Thus, when the system receives a query from the user, it can lead to two situations:

- The OA is able to complete the execution of all processes in time and delegates answers correctly to the EA.

- The OA fails to complete the execution in time and is not able to receive any answer from the PA, which means the EA shows no results to the user.

### 6.2.1 Functional Efficiency Testing

To have a better understanding of the performance of EnviroPlanner, we tested it from server-side and client-side. We conducted a performance test of the Web interface of EnviroPlanner from an external node location. It allows developers to track performance of a page or application by monitoring the Web traffic. In our system, PAs are deployed as servlets in our Tomcat server. To monitor and test the performance of the servlets, we used LoadUI Pro 2.5.5\(^1\), Smartbear’s latest load testing tool.

LoadUI Pro monitors the performance of EnviroPlanner, especially performance of the Fredericton, Moncton, and Saint John’s regions’ PAs. The servlet container contains the servlets of every PAs. The performance analysis statistics shows that system performance vary on the queries. For example, the air expert’s PA contains the `emissionNeutralizer` knowledge base which has a list of facts and rules to responses to the user query about the industrial air pollutants neutralizer. The `emissionNeutralizer` knowledge base has multiple clauses; these clauses have sub-goals/arguments and multiple facts in response to the their variables. EnviroPlanner

\(^1\)http://loadui.org/
ner performs inferences with this knowledge base to retrieve the answers for user queries about air pollutants neutralization. In contrast, the same air expert’s PA contains the *getContact* rule which consists of a few facts to respond to the user query about contact details of the head of public affairs division of ENB, Fredericton region. Based on the query delegation and rule size, the *getContact* rule has the optimum response time in comparison with *emissionNeutralizer*. The system performance analysis for the *emissionNeutralizer* query is shown in Figure 6.4. The X-axis shows the every bytes throughput of the network and the Y-axis shows the system response time. The graph shows that the servlet response time reaches its peak when a user poses this query to EnviroPlanner. Figure 6.4 shows the increase of the maximum response times for the same *emissionNeutralizer* query on doubling and tripling the number of used facts.

There are other factors that influence the performance, especially the PA performance. These include the performance of OO jDREW and arguments associated with the requested query. Table 6.1 shows several cases. The first case (the first row) shows that when there are no selected query names for the user request, the PA promptly sends “no_further_answers” message to the OA. The OA understands that there is no answer for the user request.

The EA successfully shows the RuleML answers for queries with 25 facts and 19 arguments (up to row 7 in Table 6.1). The second case shows that the PA takes different times to answer a query with 3 arguments. The first 7 rows prove that the PA slows down as the number of selected facts and arguments goes up. This happens because OO jDREW takes longer time for reasoning with more arguments.
Figure 6.2: Response Time of the 'Emission Neutralizer' Facts

Figure 6.3: Response Time After Having Doubled the Facts

Figure 6.4: Response Time After Having Tripled the Facts
Even though takes longer time for reasoning, the agents successfully answer the user query. Thus, the EA shows the RuleML answer.

Table 6.1: Running Time in Seconds

<table>
<thead>
<tr>
<th>Number of Facts</th>
<th>Number of Arguments</th>
<th>PA Performance</th>
<th>OO jDREW Performance</th>
<th>EnviroPlanner Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0.260</td>
<td>0</td>
<td>1.211</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0.282</td>
<td>0.137</td>
<td>3.781</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>3.498</td>
<td>2.810</td>
<td>7.125</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>6.010</td>
<td>4.219</td>
<td>12.921</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>14.331</td>
<td>8.161</td>
<td>20.021</td>
</tr>
<tr>
<td>21</td>
<td>12</td>
<td>18.873</td>
<td>12.381</td>
<td>25.269</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>25.873</td>
<td>19.653</td>
<td>27.718</td>
</tr>
<tr>
<td>27</td>
<td>16</td>
<td>18.873</td>
<td>12.381</td>
<td>timeout</td>
</tr>
<tr>
<td>34</td>
<td>23</td>
<td>28.873</td>
<td>24.041</td>
<td>timeout</td>
</tr>
</tbody>
</table>

Row 7 contains 19 arguments, while row 8 contains 16 arguments. Despite the smaller number of 25 facts, rather than 27 facts, the PA takes longer times to reply with an answer for the requested query in row 8 than the requested query in row 7 because because OO jDREW takes longer time for reasoning with the arguments in row 8. This happens for two reasons. The first reason is Prova take longer time for reasoning in the row 8. The second reason is because a non-trivial query with 25 facts is requested in the row 8, while a trivial query is requested in the row 7. Thus, both query types and number of facts/rules influence the performance. The last rows show the third case in which the OA fails to retrieve
the PAs RuleML message in 30 seconds. In these particular rows, the OA takes 5 minutes to obtain the RuleML query in row 9 from the EA. Moreover, the OA may run for infinitive amount of time without showing any progress on the RuleML answer obtaining process. Therefore, it has been concluded that in the worst case scenario, if OAs fail to obtain the messages from a PA within the 30 seconds time frame, Mule closed the communication channel between the OA and the EA. In that case, EnviroPlanner does not show any result but the communication channel between the OA and the PA could be remained open for further processing and interchanging operations regardless of timeout.

When timeout is exceeded, the external channels (e.g., the channel between the EA and the OA) are closed. In contrast, the internal channels (e.g., the channel between the OA and the PA) remain open for further processing and interchanging operations regardless of timeout. Based on that, if the delay is caused by processing the EA’s query, the OA submits the RuleML query to the responsible PA regardless of the timeout; if the delay is caused by processing the PA’s answer, the fully ground message remains in the temporary storage of Prova without an open channel. Prova fails to destroy this temporary storage which prevents any further processing for the next messages via the OA [22]; the OA must be restarted to accept new messages from the users. Thus, EnviroPlanner’s PAs can obtain the RuleML message whenever messages are sent by the OAs. In addition, PAs can retrieve the RuleML answer, and send it back to the OAs regardless of timeout. Increasing timeout does not help to avoid the delay because the OAs may take infinitive amount of time to obtain a single RuleML answer. However, some pos-
sible solutions are presented for improving the performance of EnviroPlanner in the Performance Improvement section of this chapter.

6.2.2 Benchmark Testing

To analyze the availability, response time, and downtime time duration from a client side, we tested our system from a remote location using Site24X7, a Web server monitoring, server monitoring, and website monitoring service of Zoho Corporation Pvt. Ltd. Figure 6.5 shows that overall, EnviroPlanner (the Super-OA level) takes more time to respond in comparison with Fredericton, Moncton, and Saint John regions (the Sub-OA levels). The Super-OA is designed to redirect queries sent by a user to the appropriate Sub-OA and receives the answers from that Sub-OA. In contrast, when a user sends queries directly to the Sub-OAs, they are delivered to the appropriate PAs and answers are returned to the users via the EA.

![Figure 6.5: Response Time Comparison Among Super-OA and Sub-OA levels](image)

Figure 6.5: Response Time Comparison Among Super-OA and Sub-OA levels
We compared EnviroPlanner with SymposiumPlanner-2011 (SP-2011) and SymposiumPlanner-2012 (SP-2012) because both of this instantiations use Rule Responder as their base framework. Using Site24X7, we tested the overall DNS time, connection time, first byte time, and total page loading time of both system. Figure 6.7 shows that EnviroPlanner has better first byte receiving time, page loading and document completion time. By performing the benchmark testing, we found that connection time and rendering times of EnviroPlanner need to be improved.

We also tested and compared some similar rule bases of SP-2011, SP-2012 and EnviroPlanner by using PSI Probe [10]. For example, we compared the viewSponsors rule base of SP-2011 and SP-2012 with the soilTest rule base of EnviroPlanner.

Table 6.2: Comparison of Performance

<table>
<thead>
<tr>
<th>Number of Queries</th>
<th>Number of Facts</th>
<th>SymposiumPlanner-2011</th>
<th>SymposiumPlanner-2012</th>
<th>EnviroPlanner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4.241</td>
<td>3.743</td>
<td>3.171</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6.572</td>
<td>4.912</td>
<td>5.215</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>6.916</td>
<td>7.864</td>
<td>7.329</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>9.258</td>
<td>8.619</td>
<td>7.952</td>
</tr>
</tbody>
</table>

Table 6.2 shows that SP-2011 takes 4.241 seconds to respond to a query, which has 4 facts, via reasoning its responsible personal agent; SP-2012 takes 3.743 seconds to respond to the same query. EnviroPlanner takes 3.171 seconds to respond a query with same number of facts and associated arguments. For multiple queries,
### Response Time Summary from California at Fri Aug 02 10:03:12 PDT 2013

<table>
<thead>
<tr>
<th>URL</th>
<th>Status</th>
<th>Size</th>
<th>Resp. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>474</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>402</td>
<td>0.10</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>272</td>
<td>0.64</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>326</td>
<td>0.64</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>164</td>
<td>0.64</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
<td>263</td>
<td>0.64</td>
</tr>
<tr>
<td>7</td>
<td>200</td>
<td>164</td>
<td>0.64</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>263</td>
<td>0.64</td>
</tr>
<tr>
<td>9</td>
<td>200</td>
<td>776</td>
<td>0.64</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
<td>354</td>
<td>0.64</td>
</tr>
<tr>
<td>11</td>
<td>200</td>
<td>339</td>
<td>0.64</td>
</tr>
<tr>
<td>12</td>
<td>200</td>
<td>164</td>
<td>0.64</td>
</tr>
<tr>
<td>13</td>
<td>200</td>
<td>263</td>
<td>0.64</td>
</tr>
<tr>
<td>14</td>
<td>200</td>
<td>734</td>
<td>1.05</td>
</tr>
</tbody>
</table>

### Response Time Summary from California at Fri Aug 02 10:38:02 PDT 2013

<table>
<thead>
<tr>
<th>URL</th>
<th>Status</th>
<th>Size</th>
<th>Resp. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>353</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>411</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>85</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>85</td>
<td>0.25</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>551</td>
<td>0.25</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
<td>318</td>
<td>0.25</td>
</tr>
<tr>
<td>7</td>
<td>200</td>
<td>915</td>
<td>0.25</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>792</td>
<td>0.25</td>
</tr>
</tbody>
</table>

---

**Figure 6.6: Analysis of EnviroPlanner**

**Figure 6.7: Analysis of SP-2012**
we estimate the average response time based on the individual response time of each query request. For example, in the second row, we tested SP-2011 and SP-2012 to get the response times for their “getContact” and “viewMediaPartners” queries individually; after getting the response times of each query request, we estimated the average response times (6.572 and 4.912 respectively) of both queries. EnviroPlanner and SP-2012 both follow SEDA style [61], which decomposes a complex, event-driven application into a set of stages connected by event queues to avoid the high overhead associated with event queues. However, although event queues decouples the execution of distributed components, it increases response time correspondingly [63]. That is why, in the third row, EnviroPlanner takes more times to response the user queries in comparison with SP-2012 because in EnviroPlanner, queries are associated with more facts and arguments than the queries of SP-2012. Last row of the Table 6.2 shows that EnviroPlanner’s performance is slightly lower than the SP-2011 and SP-2012.

In addition, it does not show any lack in performance when more rules are added to the RuleML query. For example, the answer for a query from the air expert’s profile (with 7 rules) can be obtained in 9 seconds. Adjusting the same air expert’s profile (with 28 rules), the same query’s results can be obtained in 12.536 seconds. Thus, Rule Responder is fast enough to run EnviroPlanner in a practical setting, and its performance is not influenced by the number of rulebases.
6.2.3 Prova Performance Analysis

In Prova, a fact is a rule without a body, while a goal is a rule without a head. They both may have any combination of terms (constants, variables, and lists). In our system, we implemented global knowledge bases using Prova. Because of the some limitations of Prova (e.g., lack of performance for data loading and computing time [45]), the OA creates a delay in receiving RuleML messages (queries or answers) and in sending answers back to the EA. This delay depends on the number of arguments and their data types in the message. The OA performance includes the required time for interchanging messages with other agents (i.e., EA and PA). EnviroPlanner has a 30 seconds timeout. If the OA fails to retrieve the messages from the PA within this time, the channel between the OA and the EA is closed by Mule. This means answers are not shown in the EA.

6.2.4 Scalability Testing

To test the compatibility of EnviroPlanner, we configured it on different versions of Windows machine and different versions of the Mule ESB. The test results show that average response times vary over the different versions of the Mule ESB. When EnviroPlanner in running on a updated version of Mule, then it takes less time in response to user queries which is shown in the Table 6.3.
In the case of using the latest versions of the Mule ESB, Mule 3.4, our system does not show any results to users because the upgraded endpoint configuration of Mule 3.4 is not compatible with the Rule Responder framework. EnviroPlanner can be integrated with a larger system if it has the Rule Responder framework, and the Mule ESB as the communication infrastructure. Our system is working in the Apache Tomcat server, and Java 6. It does not work with Java 5 or previous versions.

Different inference engines can be used for local and global KBs as long as they can interpret queries and answers in a common interchange language such as RuleML. For example, the OO jDREW rule engine can be used to process and obtain query answers from the local knowledge bases, and the Prova rule engine can be used to process the global knowledge bases to delegate queries to the appropriate PA. KBs of the PAs need to be compatible with the local inference engine so that it can inference with rules. Each expert’s profile can be updated by updating its knowledge base (e.g., facts and rules). A PA’s responsibilities can be extended by storing more facts and rules in its profile. The size of a knowledge base vary

<table>
<thead>
<tr>
<th>Tested Query</th>
<th>Mule Version</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3.1.0</td>
<td>0.321</td>
<td>0.670</td>
<td>0.452</td>
</tr>
<tr>
<td>8</td>
<td>3.2.1</td>
<td>0.288</td>
<td>0.642</td>
<td>0.423</td>
</tr>
<tr>
<td>8</td>
<td>3.3.1</td>
<td>0.189</td>
<td>0.113</td>
<td>0.390</td>
</tr>
<tr>
<td>8</td>
<td>3.4.0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
depending on the number of rules/facts. The size of the KBs is dictated by the desired query parameters and functionalities of the system; no absolute limits are made by the architecture or representation of EnviroPlanner. In our system, size of the KBs’ size is range from 5 kilobytes to 44 kilobytes. To determine the existence of a relationship between response time and size in KBs, we measured our KBs in kilobytes. We partitioned the KBs two groups and used t-test to check if response times are statistically significantly related to the size of the KBs.

<table>
<thead>
<tr>
<th>Range of Size</th>
<th>t-value</th>
<th>Significance</th>
<th>Avg Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-30 kilobytes</td>
<td>2.886</td>
<td>0.001</td>
<td>0.241</td>
</tr>
<tr>
<td>5-44 kilobytes</td>
<td>0.899</td>
<td>0.040</td>
<td>0.489</td>
</tr>
<tr>
<td>31-44 kilobytes</td>
<td>2.095</td>
<td>0.050</td>
<td>0.752</td>
</tr>
</tbody>
</table>

It was found that the KBs that are less than 30 kilobytes were not statistically significantly related to size but KBS that are greater than or equal to 30 kilobytes, were statistically significantly related to size (Table 6.4).

6.2.5 Performance Improvement

Our system performance is not optimized yet. By taking the following steps, the overall performance of EnviroPlanner can be improved.

- In our system, we use the Java 6 and Tomcat 6 versions, but Java 7 has better garbage collection comparison to earlier versions. Improved memory management of Tomcat 7 and Java 7 can improve the performance of EnviroPlanner.
• EnviroPlanner uses Mule ESB 3.2.1; Mule ESB 3.4 provides Mule Expression Language (MEL) for the business logic implementation with the use of expressions. It comes with an upgraded Mule studio for better support in terms of configuring connections in the GUI mode.

• Prova 3.2.1 made a significant improvements to accelerate the intra-agent message passing, which can enhance the system performance as the incoming reaction message is matched to rules inside the OA which is implemented using Prova. Reactive messaging and event processing in Prova 3.1.3 comprises a number of required parameters including a conversation-id, protocol, destination or source, performative, and payload. In the Prova 3.2.1 version, for asynchronous messaging between reactive rules within the same Prova engine, some of those parameters can be given reasonable default values, thereby achieving modest syntax simplification and speed improvement [48].
Chapter 7

Conclusions and Future Work

In this thesis, we designed and implemented a distributed environmental querying system as a prototype of a semi-automated VEO infrastructure for problem-oriented query answering, deployed as a Rule Responder use case at the url http://responder.ruleml.org/EnviroPlanner/EnviroPlanner.htm. EnviroPlanner is a prototype of a VEO infrastructure using OAs to support distributed human experts with PAs encoding some of their knowledge for answering environmental queries of EA-interfaced end users. In order to show real-world aspects of this system, we focus on New Brunswick, especially queries about the regions of Fredericton, Moncton, and Saint John, but it can be flexibly extended to further regions. We designed a Sub-OA for each region that deals with air, soil, water, and energy issues of its region. In addition, some illustrative examples of our system have been presented in the local and global knowledge bases section of chapter 5.1.2. In the following sections, discussion of the contribution of this thesis has been introduced, followed by future work.
7.1 Contributions

- We designed a OWL Lite ontology based on the four main environmental sub-domains: air, soil, water, and energy, which capture the knowledge domain of the system. The particular advantage of this OWL Lite ontology is that Prova Knowledge Base (KB) can consult with it before sending the query to the next level.

- Individual rulebases for air, soil, water, and energy queries were created and maintained by experts in each domain.

- On top of Sub-OAs, we designed a Super-OA to delegate and receive query answers to and from the local rulebases via Sub-OAs.

- We extended the functionalities of the SymposiumPlanner-2011 instantiation [52] of the Rule Responder framework, which provides architectural flexibility, and allows developers to add as many Sub-OA as they needed.

- Our system allows different rule engines to execute global and local rulebases for environmental queries. EnviroPlanner uses Prova rule engine to execute global knowledge bases and OO jDREW to execute local knowledge bases.

- An ESB has been used that carries the environmental queries, answers, and rulebases throughout the system.
7.2 Future Work

EnviroPlanner currently answers selected environmental queries related to air, soil, water, and energy domains, which should be improved, refined, and extended in collaboration with domain experts. The system can be further extended by adding additional kinds of environmental knowledge and queries, such as about storms, floods, fires, and earthquakes. Also, our current system only supports environmental queries for New Brunswick, especially Fredericton, Moncton, and Saint John. A next step would be extending EnviroPlanner so that it can answer user queries for all regions of New Brunswick and adding other provinces of Canada as well. In a Canada-wide version of EnviroPlanner, both the knowledge bases and the current architecture would need to be adjusted. One possible solution would be creating another level under the Super-OA, which would group regions from the same province. In this approach, the Super-OA would send queries to the province-level OAs, and queries would be delivered to the PAs via region-level Sub-OAs.

Currently, EnviroPlanner supports Mule ESB 3.1; hence, integrating EnviroPlanner with Mule ESB 3.4 and better Java 7 support would make it easier to deploy in different target environments. To overcome the JVM intra-agent communication limitations in previous Prova releases, Prova 3.2.1 can be implemented.
References


Appendix A

A.1 Global Knowledge Base

:-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')).
:-eval(consult('../ContractLog/owl.prova')).

performative(request):-performative(query).
performative(query).
performative(XID,Performative):-
    performative(Performative).
rcvMsg(XID, esb, From, Performative, [XI|Args]) :-
  understandPerformative(XID, From, Performative, [XI|Args]),
  processMessage(XID, From, Performative, [XI|Args]).
rcvMsg(XID, esb, From, Performative, [XI|Args]) :-
  understandPerformative(XID, From, Performative, [XI|Args]),
  rcvMsg(XID, esb, Agent, no_further_answers, Payload),
  sendMsg(XID, esb, From, no_further_answers, [XI|Args]),
  println(["----------------------------------------"]),
  println([" COMPLETE "]),
  println(["----------------------------------------"]).

interface(airQuality(ThresholdAmount, Chemical, results(Level, HealthEffects), performative(Action)), airQuality("+","+", results("-","-"), performative("-")), "Chemical alert").

understandPerformative(XID, From, "answer", Payload) :- !, fail().
understandPerformative(XID, From, "end_of_transmission", Payload) :- !, fail().
understandPerformative(XID, From, "no_further_answers", Payload) :- !, fail().
understandPerformative(XID, From, Performative, Payload) :-
  performative(Performative).
understandPerformative(XID, From, Performative, Payload) :-
  not(performative(Performative)),

133
sendMsg(XID,esb,From,"answer", notUnderstood("performative",Performative)),
sendMsg(XID,esb,From,"no_further_answers", Payload), fail().

processMessage(XID,From,Primitive,Payload):-
  rcvMsg(XID,esb, To, "no_further_answers", Answer),
  sendMsg(XID,esb,From,"no_further_answers", Payload),
  fail().

processMessage(XID,From,Primitive,[Function|Arguments]):-
  concat([ep_,Function],FunctionwithNamespace),
  assigned(XID,FunctionwithNamespace,"ep_responsible",Agent,Result),
  CandidateSubOAs=ws.prova.mule.impl.DecisionCriteria.subOADecision (Result,Function,Arguments),
  element(CandidateSubOA,CandidateSubOAs),
  sendMsg(XID,esb,CandidateSubOA, "query", [Function|Arguments]),
  rcvMult(XID,esb,CandidateSubOA, "answer", Answer),
  sendMsg(XID,esb, From, "answer", Answer).

import("http://localhost:8080/EnviroPlanner/EnviroPlanner.owl").
reasoner("owl").

agent(XID,Agent:ep_FunctionManagement).
agent(XID,Agent):-
    sendMsg(XID,esb,"EnviroPlanner","no_further_answers",
    agent(Member)),fail().

responsibility(XID,Domain:ep_OAManagement).
role(XID,Role:owl_ObjectProperty).

% responsibility assignment matrix
assigned(XID,Responsibility,Role,Agent,Result):-
    import(URL),
    reasoner(Reasoner),
    rdf(URL,Reasoner,Responsibility,Role,Agent,Result).

A.2 Experts’ Profile Examples

A.2.1 Fredericton Region Air

chemInterval2LevelHealthFive(CarbonMonoxide:string,
    interval[ppb[76:integer], ppb[100:integer]], Poor,
    healthEffects["Increasing symptoms in nonsmokers
    with heart disease; blurred vision; some clumsiness":string]).

chemInterval2LevelHealthFour(CarbonMonoxide:string,
    interval[ppb[46:integer], ppb[75:integer]], Moderate,
    healthEffects["Increased symptoms in smokers
with heart disease":string])

chemInterval2LevelHealthThree(CarbonMonoxide:string, interval[ppb[26:integer], ppb[45:integer]], Good, healthEffects["Blood chemistry changes but no noticeable impairment":string])

chemInterval2LevelHealthTwo(CarbonMonoxide:string, interval[ppb[11:integer], ppb[25:integer]], VeryGood, healthEffects["No health effects are expected in healthy people":string])

chemInterval2LevelHealthOne(CarbonMonoxide:string, interval[ppb[0:integer], ppb[10:integer]], Excellent, healthEffects["No health effects are expected in healthy people":string])

chemInterval2LevelHealthFive(NitrogenDioxide:string, interval[ppb[81:integer], ppb[100:integer]], Poor, healthEffects["Increasing sensitivity for asthmatics and people with bronchitis":string])

chemInterval2LevelHealthFour(NitrogenDioxide:string, interval[ppb[51:integer], ppb[80:integer]], Moderate,
healthEffects["Air smells and looks brown.
Some increase in bronchial reactivity in asthmatics"].

chemInterval2LevelHealthThree(NitrogenDioxide, interval[ppb[31:integer], ppb[50:integer]], Good, healthEffects["Odour"].)

chemInterval2LevelHealthTwo(NitrogenDioxide, interval[ppb[16:integer], ppb[30:integer]], VeryGood, healthEffects["Slight odor"].)

chemInterval2LevelHealthOne(NitrogenDioxide, interval[ppb[0:integer], ppb[15:integer]], Excellent, healthEffects["No health effects are expected in healthy people"].)

chemInterval2LevelHealthFive(?Chemical, interval[ppb[?MinPoorAmount:integer], ppb[?MaxPoorAmount:integer]], ?Level, ?HealthEffects), greaterThanOrEqual(?ThresholdAmount, ?MinPoorAmount),
lessThanOrEqualTo(?ThresholdAmount:integer, ?MaxPoorAmount:integer).

    chemInterval2LevelHealthFour(?Chemical:string, interval[ppb[?MinModerateAmount:integer], ppb[?MaxModerateAmount:integer]], ?Level, ?HealthEffects),
    greaterThanOrEqualTo(?ThresholdAmount:integer, ?MinModerateAmount:integer),
    lessThanOrEqualTo(?ThresholdAmount:integer, ?MaxModerateAmount:integer).

    chemInterval2LevelHealthThree(?Chemical:string, interval[ppb[?MinGoodAmount:integer], ppb[?MaxGoodAmount:integer]], ?Level, ?HealthEffects),
    greaterThanOrEqualTo(?ThresholdAmount:integer, ?MinGoodAmount:integer),
    lessThanOrEqualTo(?ThresholdAmount:integer, ?MaxGoodAmount:integer).

chemInterval2LevelHealthTwo(?Chemical:string, interval[ppb[?MinVeryGoodAmount:integer], ppb[?MaxVeryGoodAmount:integer]], ?Level, ?HealthEffects),
greaterThanOrEqual(?ThresholdAmount:integer, ?MinVeryGoodAmount:integer),

chemInterval2LevelHealthOne(?Chemical:string, interval[ppb[?MinExcellentAmount:integer], ppb[?MaxExcellentAmount:integer]], ?Level, ?HealthEffects),
greaterThanOrEqual(?ThresholdAmount:integer, ?MinExcellentAmount:integer),
lessThanOrEqual(?ThresholdAmount:integer, ?MaxExcellentAmount:integer).

level2Message(Poor, CarbonMonoxide:string, HighAlert).
level2Message(Moderate, CarbonMonoxide:string, HighAlert).
level2Message(Good, CarbonMonoxide:string, Alert).
level2Message(VeryGood, CarbonMonoxide:string, NoAlert).
level2Message(Excellent, CarbonMonoxide:string, NoAlert).
level2Message(Poor, NitrogenDioxide:string, HighAlert).
level2Message(Moderate, NitrogenDioxide:string, Alert).
level2Message(Good, NitrogenDioxide:string, Alert).
level2Message(VeryGood, NitrogenDioxide:string, NoAlert).
level2Message(Excellent,NitrogenDioxide:string, NoAlert).

airQuality(?ThresholdAmount:integer, ?Chemical:string,
    results[?Level, ?HealthEffects],performative[?AlertMessage]) :-
    chemAlert(?ThresholdAmount:integer, ?Chemical:string,
    ?Level, ?HealthEffects),

airPollutantEmission(?Year:integer, ?CompanyName:string,
    ?FacilityName, ?UnitofMeasure,
    greenHouseGas[?TotalGHGEmission, ?AmountofCO2,
    ?AmountofCH4, ?AmountofN20,
    ?AmountofOtherGHGases], ?TotalEmission) :-
    source(?CompanyName:string, ?FacilityName,
    gHGEmission(?FacilityName, ?Year:integer, ?AmountofCO2,
    ?AmountofCH4, ?AmountofN20, ?AmountofOtherGHGases:integer),
    add(?TotalGHGEmission:integer, ?AmountofCO2:integer,
    ?AmountofCH4:integer, ?AmountofN20:integer,
?AmountofOtherGHGases:integer),

source("ABC Corporation":string, facilityName["ABC Petroleum Limited":string], 2012:integer, Ton, 90:integer).
gHGEmission(facilityName["ABC Petroleum Limited":string], 2012:integer, 8:integer, 6:integer, 5:integer, 15:integer).

source("ABC Corporation":string, facilityName["ABC Pulp and Paper Mill":string], 2012:integer, Ton, 70:integer).

source("XYZ Inc.":string, facilityName["XYZ Chemical Industry":string], 2011:integer, Ton, 100:integer).
gHGEmission(facilityName["XYZ Chemical Industry":string], 2011:integer, 10:integer, 6:integer, 6:integer, 20:integer).

source("XYZ Inc.":string, facilityName["XYZ Papers Company Limited":string], 2011:integer, Ton, 80:integer).
heatAlert(?HumidexValue:integer, alert[?Level, ?DegereeOfComfort, ?Effects]) :-
    alertLevel(?HumidexValue:integer, ?Level),

level(range[value[0:integer], value[29:integer]], levelOne).
level(range[value[30:integer], value[39:integer]], levelTwo).
level(range[value[40:integer], value[45:integer]], levelThree).

alertLevel(?HumidexValue:integer, levelFour) :-
    greaterThanOrEqual(?HumidexValue:integer, 46:integer).

alertLevel(?HumidexValue:integer, ?Level) :-
    lessThanOrEqual(?HumidexValue:integer, 45:integer),
    testLevel(?HumidexValue:integer, ?Level).

testLevel(?HumidexValue:integer, levelOne) :-
    greaterThanOrEqual(?HumidexValue:integer, 0:integer),
    level(range[value[?MinLevelOne:integer],
    value[?MaxLevelOne:integer]], levelOne),
    lessThanOrEqual(?HumidexValue:integer, ?MaxLevelOne:integer).

testLevel(?HumidexValue:integer, levelTwo) :-
greaterThanOrEqual(?HumidexValue:integer, 30:integer),
level(range[value[?MinLevelTwo:integer],
value[?MaxLevelTwo:integer]], levelTwo),
lessThanOrEqual(?HumidexValue:integer, ?MaxLevelTwo:integer).

testLevel(?HumidexValue:integer, levelThree) :-
greaterThanOrEqual(?HumidexValue:integer, 40:integer),
level(range[value[?MinLevelThree:integer],
value[?MaxLevelThree:integer]], levelThree),
lessThanOrEqual(?HumidexValue:integer, ?MaxLevelThree:integer).

comfort(levelOne, Comfortable, effects["No effects":string]).
comfort(levelTwo, SomeDiscomfort, effects["The main
cause of illness during a heat wave is the aggravation of
preexisting respiratory and cardiovascular diseases":string]).
comfort(levelThree, GreatDiscomfort, effects["Everyone is at
risk of heat stress and heat stroke":string]).
comfort(levelFour, Dangerous, effects["Everyone is at high
risk for heat related illnesses and heat stroke":string]).

emissionNeutralizer(?Factory:string, ?Pollutant, ?Remedy,
?ChemCompound, measurement[?AmountofPollutant,
?AmountofRemedy, ?UnitofMeasurement]) :-
  multiply(?AmountofRemedy:real, ?AmountofPollutant:real, 0.20:real).

emit("AB Chemical Limited":string, 05:real, PhosphoricAcid).
emit("AB Chemical Limited":string, 08:real, HydrobromicAcid).
emit("AB Chemical Limited":string, 07:real, NitricAcid).
emit("AB Chemical Limited":string, 14:real, CarbonDioxide).
emit("AB Chemical Limited":string, 18:real, SulfuricAcid).
emit("NB Pulp and Paper Limited":string, 10:real, SulfuricAcid).
emit("NB Pulp and Paper Limited":string, 02:real, HydrobromicAcid).
emit("NB Pulp and Paper Limited":string, 05:real, Phosphoric Acid).

chemReact(NitricAcid, SodiumHydroxide, SodiumNitrate, Ton).
chemReact(NitricAcid, PotassiumHydroxide, PotassiumNitrate, Ton).
chemReact(CarbonDioxide, PotassiumHydroxide, PotassiumBicarbonate, Ton).
chemReact(CarbonDioxide, LithiumHydroxide, LithiumCarbonate, Ton).
chemReact(SulfuricAcid, SodiumHydroxide, SodiumSulphate, Ton).
chemReact(CarbonicAcid, MagnesiumHydroxide, MagnesiumCarbonate, Ton).


city(?CityName:string, ?ToleranceLevel, ?Tolerability),


localTol(?CityName:string, ?LocalC02Tolerance:integer, ?LocalNO2Tolerance:integer, ?LocalNH4Tolerance:integer),
lessThan0rEqual(?C02Tolerance:integer, ?LocalC02Tolerance:integer),
lessThanDrEqual(?NO2Tolerance:integer, ?LocalNO2Tolerance:integer),
lessThan0rEqual(?NH4Tolerance:integer, ?LocalNH4Tolerance:integer).


city(?CityName:string, ?ToleranceLevel, ?Tolerability),
?NH4Tolerance:integer, Good) :-
  localTol(?CityName:string, ?LocalCO2Tolerance:integer,
  ?LocalNO2Tolerance:integer, ?LocalNH4Tolerance:integer),
  greaterThan(?CO2Tolerance:integer,
  ?LocalCO2Tolerance:integer),
  greaterThan(?NO2Tolerance:integer,
  ?LocalNO2Tolerance:integer),
  greaterThan(?NH4Tolerance:integer,
  ?LocalNH4Tolerance:integer).

city("City A":string, Risky, NotLiveable).

city("City A":string, Good, Liveable).

localTol("City A":string, 1:integer, 2:integer, 1:integer).

A.2.2 Fredericton Region Energy

powerStation(?StationName, ?InstalledCapacity, ?StationType,
  ?HydraulicHead, ?Turbines, ?TypeofDam, ?Height,
  ?Length, ?NumofSpillways) :-
  station(?StationName, ?InstalledCapacity, ?StationType,
  ?HydraulicHead, ?Turbines),
  dam(?Height, ?Length, ?NumofSpillways,
  ?TypeofDam, ?StationName).

station(stationName["Mactaquac Generating Station":string],

146
installedCapacity["653MW":string], stationType["Run-of-the-River":string],
ydraulicHead["31.7 meter to 35.4 meter":string],
turbines["6 X Kaplan":string]).
dam(height["55 meter":string], length["518 meter":string],
numofSpillways["2 Concrete Spillways":string],
typeofDam["Embankment":string],
stationName["Mactaquac Generating Station":string]).

energyServices(?ServiceProvider, ?ConsultancyServices,
?RentalServices, ?RentalServicePenalty, ?ContactInfo) :-
  provider(?ServiceProvider, ?ConsultancyServices, ?RentalServices),
  monthlyPenalty(?RentalServicePenalty, ?ContactInfo, ?ServiceProvider).

provider(serviceProvider["Fredericton city council":string],
  consultancyServices["Electrical utility":string,
  "Green energy advice":string,
  "Commercial and residential energy advice":string,
  "Street light repair":string, "Energy efficient home
  up-gradation program":string,
  "Load program":string, "Power outage info":string],
rentalServices["Water heater rental":string,
  "Lighting rental":string]).

monthlyPenalty(rentalServicePenalty["1.5 Percent of monthly bill":string],
  contactInfo["111 king street":string,"Fredericton,
serviceProvider["Fredericton city council":string],

provider(serviceProvider["ABC Energy":string],
    consultancyServices[ "Commercial energy consultation":string],
    rentalServices["Water heater rental":string,"Lighting rental":string]).

monthlyPenalty(rentalServicePenalty["2.0 Percent of monthly bill":string],
    contactInfo["123 Regent street":string,
        "Fredericton, NB":string,
        "E3B1X2":string, "phone 5061234123":string],
    serviceProvider["ABC Energy":string]).

A.2.3 Fredericton Region Soil

soilTest(?SoilSample, fees[?WormTestFee, ?RecordTestFee, ?PHTestFee],
    ?TestCenterAddress) :-
    soil(?SoilSample, ?Address, ?WormTestFee),
    fees(?RecordTestFee, ?PHTestFee, ?SoilSample).

soil(LawnGardening, address["XYZ Soil Testing":string,
    "Fredericton, NB, E3B5H1":string,
    "Contact number 5061234123":string], dollar[10]).
fees(dollar[50], dollar[8], LawnGardening).

soil(FarmField, address["ABC Agricultural Lab":string,
soilPollutantRelease(?Year:integer, ?CompanyName:string, ?FacilityName, ?AmountofDumpedTrash,
  ?TotalRelease, ?MajorSubstances) :-
  source(?CompanyName:string, ?FacilityName, ?Year:integer, ?TotalRelease),

source("XYZ Inc.":string, facilityName["XYZ Pulp and Papers":string], 2012:integer, ton[10]).
substances(facilityName["XYZ Pulp and Papers":string], 2012:integer, majorSubstances["Hydrogen Peroxide, Magnesium Silicate, Sulphuric Acid, Sodium Carbonate, Alganic acid, Sodium Hydroxide":string], ton[1.5]).

source("XYZ Inc.":string, facilityName["XYZ Pulp and Papers":string], 2011:integer, ton[12]).
substances(facilityName["XYZ Pulp and Papers":string], 2011:integer, majorSubstances["Hydrogen Peroxide, Magnesium Silicate, Sulphuric Acid,
Sodium Carbonate, Alganic acid, Sodium Hydroxide: string, ton[2]).

A.2.4 Fredericton Region Water

alterationPermit(?PermitType, ?IntendedWork, ?PermitFee, ?YearlyRenewalFee) :-
    permit(?PermitType, ?PermitFee, ?YearlyRenewalFee),
    purpose(?IntendedWork, ?PermitType).

%%StandardPermit permit

permit(StandardPermit, dollar[30], dollar[10]).
purposes(intendedWork["Engineering design; installing or modifying a dam or other water level control structure; or installing or modifying a pipeline crossing": string,
                      "Engineering investigation": string,
                      "Industrial installation": string], StandardPermit).

% Provisional permit

permit(ProvisionalPermit, $20, $8).
purpose(intendedWork["Land scaping": string,
                       "Removing vegetation from a wetland or the bank or bed of a watercourse": string], ProvisionalPermit).

waterShedZone(?ZoneType:string, shedProtection[?WaterShedName, ?Boundary, ?PermittedActivities]) :-
drinkingwaterShed(?ZoneType:string, ?WaterShedName, ?Boundary),
activities(?PermittedActivities, ?ZoneType:string).

%permitted activities in the shed protection zones A
drinkingwaterShed(A:string, waterShedName["Watershed of
unnamed tributary to Saint John river":string],
boundary["The watercourse":string]).
activities(permtrustedActivities["Undertake boating and
fishing in non-motorized watercraft":string,
"Undertake surveying or sign posting":string], A:string).

%permitted activities in the shed protection zones B
drinkingwaterShed(B:string, waterShedName["Watershed of
unnamed tributary to Saint John river":string], boundary["The
75-meter setback from the water course":string]).
activities(permtrustedActivities["carry out emergency
operations conducted for public health; safety or
general welfare":string,
"access for primitive recreation uses;
i.e., study of wildlife,
hunting, fishing, trapping, canoeing and canoe portaging,
cross-country skiing and snowshoeing":string,
"operate a motorized vehicle on an existing
provincial highway":string,
    "survey and post signs":string], B:string).

%permitted activities in the shed protection zones C
drinkingwaterShed(C:string, waterShedName["Watershed of
unnamed tributary to Saint John river":string], boundary["The
balance of the watershed area":string]).
activities(permittedActivities["Construct, use, maintain,
renovate, make additions to or rebuild a family dwelling
and any accessory buildings and structures":string,
    "Carry out tree planting activities":string,
    "Carry out forestry practices":string,
    "Carry out existing agricultural activities":string,
    "Carry out the extraction and crushing of sand,
gravel and other aggregate materials":string], C:string).

waterPollutantRelease(?Year:integer, ?CompanyName:string,
    ?FacilityName, ?DirectDischarge,
    ?TotalRelease, ?MajorSubstances) :-
    source(?CompanyName:string, ?FacilityName,
    ?Year:integer, ?TotalRelease),
    substances(?FacilityName, ?Year:integer,
source("ABC Corporation":string, facilityName["ABC Chemical Fredericton":string], 2012:integer, ton[12]).

substances(facilityName["ABC Chemical Fredericton":string], 2012:integer,
    majorSubstances["Aluminum, Calcium, Potassium, Magnesium, Sodium":string], ton[3]).

A.3 EnviroPlanner Role Assignment Matrix

<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [ 
  <!ENTITY epREGIONATfredericton 'http://ibis.in.tum.de/projects/paw/2011#'> 
  <!ENTITY epREGIONATsaintJohn 'http://ibis.in.tum.de/projects/paw/2010#'> 
  <!ENTITY epREGIONATmoncton 'http://ibis.in.tum.de/projects/paw/2010#'> 
  <!ENTITY epREGIONATenviroPlanner 'http://ibis.in.tum.de/projects/paw/2012#'> ]>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:ep="http://ibis.in.tum.de/projects/paw#"
xmlns:epREGIONATfredericton="&epREGIONATfredericton;"
<owl:Ontology rdf:about="./EnviroPlanner.owl">
  <owl:versionInfo>v 0.01</owl:versionInfo>
  <rdfs:comment>Describes the responsibility assignment matrix of EnviroPlannerSystem</rdfs:comment>
</owl:Ontology>

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

<!-- Model of the EnviroPlanner NameSpace Management -->
<owl:Class rdf:ID="FunctionManagement”>
  <rdfs:subClassOf rdf:resource="#EnviroPlanner" />
</owl:Class>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

157
<owl:Class rdf:ID="powerStation">
  <rdfs:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

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

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

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

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

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

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

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

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

<!-- Assign Responsibilities to organizing committee members -->
<owl:ObjectProperty rdf:ID="responsible">
<rdf:type rdf:resource="owl#FunctionalProperty" />
<rdfs:domain rdf:resource="#FunctionManagement" />
<rdfs:range rdf:resource="&epREGIONATfredericton;EP-REGION-FREDERICTON"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="responsible">
<rdf:type rdf:resource="owl#FunctionalProperty" />
</owl:ObjectProperty>
<rdfs:domain rdf:resource="#FunctionManagement" />  
<rdfs:range rdf:resource="&epREGIONATsaintJohn;  
EP-REGION-SAINTJOHN"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="responsible">
<rdf:type rdf:resource="owl#FunctionalProperty" />  
<rdfs:domain rdf:resource="#FunctionManagement" />  
<rdfs:range rdf:resource="&epREGIONATmoncton;  
EP-REGION-MONCTON"/>
</owl:ObjectProperty>

<airQuality rdf:ID="airQuality">
<responsible rdf:resource="&epREGIONATfredericton;  
EP-REGION-FREDERICTON" /></airQuality>

<airPollutantEmission rdf:ID="airPollutantEmission">
<!-- <responsible rdf:resource="&epREGIONATfredericton;  
EP-REGION-FREDERICTON"/>  -->
<!-- <responsible rdf:resource="&epREGIONATsaintJohn;  
EP-REGION-SAINTJOHN"/>  -->
<responsible rdf:resource="&epREGIONATmoncton;  
EP-REGION-MONCTON" /></airPollutantEmission>
<chemTolerance rdf:ID="chemTolerance">
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON"/>
</chemTolerance>

<emissionNeutralizer rdf:ID="emissionNeutralizer">
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON"/>
</emissionNeutralizer>

<waterPollutantRelease rdf:ID="waterPollutantRelease">
    <responsible rdf:resource="&epREGIONATsaintJohn; EP-REGION-SAINTJOHN"/>
</waterPollutantRelease>

<soilPollutantRelease rdf:ID="soilPollutantRelease">
    <responsible rdf:resource="&epREGIONATsaintJohn; EP-REGION-SAINTJOHN"/>
</soilPollutantRelease>

<energyServices rdf:ID="energyServices">
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON"/>
</energyServices>
<responsible rdf:resource="&epREGIONATsaintJohn;
    EP-REGION-SAINTJOHN" />
</energyServices>

<waterHeater rdf:ID="waterHeater">
    <responsible rdf:resource="&epREGIONATfredericton;
        EP-REGION-FREDERICTON" />
    <responsible rdf:resource="&epREGIONATsaintJohn;
        EP-REGION-SAINTJOHN" />
</waterHeater>

<alterationPermit rdf:ID="alterationPermit">
    <responsible rdf:resource="&epREGIONATsaintJohn;
        EP-REGION-SAINTJOHN" />
    <responsible rdf:resource="&epREGIONATfredericton;
        EP-REGION-FREDERICTON" />
</alterationPermit>

<rentalWaterHeater rdf:ID="rentalWaterHeater">
    <responsible rdf:resource="&epREGIONATsaintJohn;
        EP-REGION-SAINTJOHN" />
    <responsible rdf:resource="&epREGIONATfredericton;
        EP-REGION-FREDERICTON" />
</rentalWaterHeater>
<waterShedZone rdf:ID="waterShedZone">
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
</waterShedZone>

<soilTest rdf:ID="soilTest">
    <responsible rdf:resource="&epREGIONATsaaintJohn; EP-REGION-SAINTJOHN" />
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
    <responsible rdf:resource="&epREGIONATmoncton; EP-REGION-MONCTON" />
</soilTest>

<sourcesOfPollutants rdf:ID="sourcesOfPollutants">
    <responsible rdf:resource="&epREGIONATsaaintJohn; EP-REGION-SAINTJOHN" />
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
    <responsible rdf:resource="&epREGIONATmoncton; EP-REGION-MONCTON" />
</sourcesOfPollutants>
<getExpertContact rdf:ID="getExpertContact">
    <responsible rdf:resource="&epREGIONATsa;ntJohn;" EP-REGION-SAINTJOHN" />
    <responsible rdf:resource="&epREGIONATfred;ericton;" EP-REGION-FREDERICTON" />
    <responsible rdf:resource="&epREGIONATmoncton;" EP-REGION-MONCTON" />
</getExpertContact>

<nuclearPowerStation rdf:ID="nuclearPowerStation">
    <responsible rdf:resource="&epREGIONATsa;ntJohn;" EP-REGION-SAINTJOHN" />
</nuclearPowerStation>

<powerStation rdf:ID="powerStation">
    <!-- <responsible rdf:resource="&epREGIONATsa;ntJohn;" EP-REGION-SAINTJOHN" -->
    <responsible rdf:resource="&epREGIONATfred;ericton;" EP-REGION-FREDERICTON" />
    <!-- <responsible rdf:resource="&epREGIONATmoncton;" EP-REGION-MONCTON" -->
</powerStation>

<heatAlert rdf:ID="heatAlert"/>
<responsible rdf:resource="&epREGIONATfredericton;
   EP-REGION-FREDERICTON" />

</heatAlert>

<riverRestoration rdf:ID="riverRestoration">
   <responsible rdf:resource="&epREGIONATmoncton;
   EP-REGION-MONCTON" />
</riverRestoration>

<waterServiceRates rdf:ID="waterServiceRates">
   <responsible rdf:resource="&epREGIONAtsaintJohn;
   EP-REGION-SAINTJOHN" />
</waterServiceRates>

<MonctonLightingRental rdf:ID="monctonLightingRental">
   <responsible rdf:resource="&epREGIONAtsaintJohn;
   EP-REGION-SAINTJOHN" />
</MonctonLightingRental>

<Tidalbore rdf:ID="tidalbore">
   <responsible rdf:resource="&epREGIONATmoncton;
   EP-REGION-MONCTON" />
</Tidalbore>
</rdf:RDF>
<owl:Class rdf:ID="soilPollutantRelease">
<rdfs:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

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

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

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

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

<owl:Class rdf:ID="drinkingWaterQuality">
<rdfs:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>
<owl:Class rdf:ID="dwServiceRate">
<o:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

<owl:Class rdf:ID="alterationPermit">
<o:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

<owl:Class rdf:ID="rentalWaterHeater">
<o:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

<owl:Class rdf:ID="waterShedZone">
<o:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

<owl:Class rdf:ID="wellField">
<o:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

<owl:Class rdf:ID="soilTest">
<o:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>
<owl:Class rdf:ID="sourcesOfPollutants">
<rdfs:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

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

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

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

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

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

168
<owl:Class rdf:ID="riverRestoration">
  <rdfs:subClassOf rdf:resource="#FunctionManagement" />
</owl:Class>

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

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

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

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

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

<!-- Assign Responsibilities to organizing committee members -->
<owl:ObjectProperty rdf:ID="responsible">
  <rdf:type rdf:resource="owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#FunctionManagement"/>
  <rdfs:range rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="responsible">
  <rdf:type rdf:resource="owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#FunctionManagement"/>
  <rdfs:range rdf:resource="&epREGIONATsaintJohn; EP-REGION-SAINTJOHN"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="responsible">
  <rdf:type rdf:resource="owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#FunctionManagement"/>
  <rdfs:range rdf:resource="&epREGIONATmoncton; EP-REGION-MONCTON"/>
</owl:ObjectProperty>
Individual agents (individuals)
Define a Responsibility Assignment Matrix-->

<airQuality rdf:ID="airQuality">
  <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
</airQuality>

<airPollutantEmission rdf:ID="airPollutantEmission">
  <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
  <responsible rdf:resource="&epREGIONATsaintJohn; EP-REGION-SAINTJOHN" />
  <responsible rdf:resource="&epREGIONATmoncton; EP-REGION-MONCTON" />
</airPollutantEmission>

<chemTolerance rdf:ID="chemTolerance">
  <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
</chemTolerance>
<emissionNeutralizer rdf:ID="emissionNeutralizer">
  <responsible rdf:resource="&epREGIONATfredericton;
  EP-REGION-FREDERICTON"/>
</emissionNeutralizer>

<waterPollutantRelease rdf:ID="waterPollutantRelease">
  <!-- <responsible rdf:resource="&epREGIONATfredericton;
  EP-REGION-FREDERICTON"/> -->
  <responsible rdf:resource="&epREGIONATsaintJohn;
  EP-REGION-SAINTJOHN" />
</waterPollutantRelease>

<soilPollutantRelease rdf:ID="soilPollutantRelease">
  <responsible rdf:resource="&epREGIONATsaintJohn;
  EP-REGION-SAINTJOHN" />
</soilPollutantRelease>

<energyServices rdf:ID="energyServices">
  <responsible rdf:resource="&epREGIONATfredericton;
  EP-REGION-FREDERICTON" />
  <responsible rdf:resource="&epREGIONATsaintJohn;
  EP-REGION-SAINTJOHN" />
</energyServices>
<waterHeater rdf:ID="waterHeater">
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
    <responsible rdf:resource="&epREGIONATsaintJohn; EP-REGION-SAINTJOHN" />
</waterHeater>

<alterationPermit rdf:ID="alterationPermit">
    <responsible rdf:resource="&epREGIONATsaintJohn; EP-REGION-SAINTJOHN" />
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
</alterationPermit>

<rentalWaterHeater rdf:ID="rentalWaterHeater">
    <responsible rdf:resource="&epREGIONATsaintJohn; EP-REGION-SAINTJOHN" />
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
</rentalWaterHeater>

<waterShedZone rdf:ID="waterShedZone">
    <responsible rdf:resource="&epREGIONATfredericton; EP-REGION-FREDERICTON" />
</waterShedZone>
<nuclearPowerStation rdf:ID="nuclearPowerStation">
  <responsible rdf:resource="&epREGIONATsaintJohn;"
    EP-REGION-SAINTJOHN" />
</nuclearPowerStation>

<powerStation rdf:ID="powerStation">
  <!-- <responsible rdf:resource="&epREGIONATsaintJohn;"
    EP-REGION-SAINTJOHN" -->
  <responsible rdf:resource="&epREGIONATfredericton;"
    EP-REGION-FREDERICTON" />
  <!-- <responsible rdf:resource="&epREGIONATmoncton;"
    EP-REGION-MONCTON" -->
</powerStation>

<heatAlert rdf:ID="heatAlert">
  <responsible rdf:resource="&epREGIONATfredericton;"
    EP-REGION-FREDERICTON" />
</heatAlert>
<riverRestoration rdf:ID="riverRestoration">
  <responsible rdf:resource="&epREGIONATmoncton;
    EP-REGION-MONCTON" />
</riverRestoration>

<waterServiceRates rdf:ID="waterServiceRates">
  <responsible rdf:resource="&epREGIONATsaintJohn;
    EP-REGION-SAINTJOHN" />
</waterServiceRates>

<MonctonLightingRental rdf:ID="monctonLightingRental">
  <responsible rdf:resource="&epREGIONATsaintJohn;
    EP-REGION-SAINTJOHN" />
</MonctonLightingRental>

<Tidalbore rdf:ID="tidalbore">
  <responsible rdf:resource="&epREGIONATmoncton;
    EP-REGION-MONCTON" />
</Tidalbore>
</rdf:RDF>
Glossary

Performative
EnviroPlanner is a distributed rule-based system where each rule agent can use a rule engine to access different sources of data. These distributed agents connect and communicate with other agents using a common rule interchange language which carries pragmatic performatives. These performatives can be used by the agents to understand the pragmatic context of the message.

Reaction RuleML
Reaction RuleML is a general, compact and user-friendly XML-serialized sublanguage of RuleML. Reaction RuleML is our interchange language between agents.

OO jDREW
OO jDREW, the Object Oriented Java Deductive Reasoning Engine for the Web is the reference implementation of the (Naf Hornlog) RuleML Web rule language. It is an Object Oriented extension to jDREW.
Prolog

Prolog is a general-purpose logic programming language based on (Horn) logic. In Prolog, program logic is expressed in terms of relations, and a computation is initiated by running a query over these relations.

Prova

Prova combines Prolog with Java; it is a Semantic Web rule language and a highly expressive rule engine with the same name, which supports complex rule-based work-flows, complex event processing, distributed inference services, rule interchange, and rule-based decision logic.

Role

A role is a descriptor of an associated set of tasks. For example, alert and pollutant topics are mapped to the role ‘air’.

Topic

Topics are mapped to a role and they can be mapped into the same address of an agent.

RAM

Rule Responder framework uses a RAM, which is implemented in Web Ontology Language, to support the OA in its selection of PAs. The built-in rdf() is used for reasoning in the OWL ontology and retrieving the responsible agent’s name from the RAM.
OWL
Web Ontology Language is one of the knowledge representation languages that is used for authoring ontologies by providing additional vocabularies along with a formal semantics.

FOAF
Friend of a Friend is a machine-readable vocabulary that is used to describe people, the links between them, their activities and relations to other people, objects, and activities.

VO
Virtual Organization is a multi-agent knowledge-based system that enables coordinations among various heterogeneous agents/services, where different rules and ontologies can be executed by different agents. Individuals and institutions/Organizations can use VOs to coordinate resources and services across institutional boundaries.

POSL
Positional-Slotted Language is a logic programming language that integrates Prolog’s positional and F-logic’s slotted syntaxes for representing knowledge (facts and rules) on the Semantic Web.
RDF
Resource Description Framework provides a common framework for representing information about Web resources. RDF information can be processed by applications rather than only being displayed to users.

RDFS
Resource Description Framework Schema extends the RDF vocabulary that allows describing the taxonomies of classes and properties of a domain.

DNS Time
The time it takes for the browser to translate a host-name to its corresponding IP address.

Connection Time
The time that it takes to connect to our server, in our case it’s Tomcat, across a network.

First Byte Time
The time interval between the time at which a request is made from a web server and the time at which the first byte of data requested is about to be transferred.
Vita

Candidate’s full name:
Sujan Chandra Saha

Universities attended (with dates and degrees obtained):
Multimedia University, Melaka, Malaysia, 2005-2008
Bachelor of Information Science and Technology

University of New Brunswick, Fredericton, Canada, 2010-2013
Masters of Computer Science

Conference Presentations:
http://www.awoss.org/awoss3/slides/Preseantion_Sujan.pdf

S. Saha, H. Boley, G. Dueck, Research Expo poster, University of New Brunswick, April 2011, “EnviroPlanner: A Rule Responder for Distributed Environmental Querying”

Professional Activities:
IT analyst, Department of Supply and Services, Government of New Brunswick
January 2011 August 2011, Fredericton, NB

Tech Pub intern, Genesys Telecommunication Lab
August 2012 December 2012, Saint John, NB
Projects: