Project Report

Extending OO jDREW for RuleML 1.0

by

Omar Alsaiari, Khalid F. AlMutariri, Nasser Albunian,
Christian Fabbricatore, Markus Zucker

omar.alsaiari@gmail.com, k.almutairi@hotmail.com,
{nass11.s, c.fabbricatore, m.zucker}@unb.ca

Course: Semantic Web Techniques (CS6795)
Instructor: Harold Boley
Advisors: Bruce Spencer, Tara Athan, Ben Craig
Term: Fall 2011
Date: January 12, 2012
# Contents

## 1 Introduction
1.1 Issues ........................................ 2  
1.2 Objectives .................................... 3  
1.3 Approach ..................................... 4  
1.4 Tools ......................................... 5  

## 2 General & Miscellaneous Improvements
2.1 Transition to GitHub .......................... 6  
2.2 Reorganization of the Project Structure  .... 7  
2.3 Re-Engineering Effort ....................... 8  
2.4 Transition to Maven ........................... 11  
2.4.1 Removal of the custom XML Serialization Routines  ... 12  
2.5 POSL Parser Update ......................... 12  
2.6 JAR Signing .................................. 12  

## 3 RuleML Parser
3.1 Source Code Analysis ....................... 16  
3.1.1 Internal Data Structures of the RuleML Parser  ... 18  
3.2 Difference between RuleML 1.0 and RuleML 0.91 .... 19  
3.3 Derived Objectives ......................... 19
List of Figures

2.1 Packet Structure of OO jDREW ................................. 8
3.1 RuleML 0.88 and 0.91 parser infrastructure in OO jDREW 0.96I ........................ 17
3.2 RuleML 1.0 striped vs. striped-skipped syntax ............................. 17
3.3 New RuleML parser infrastructure in OO jDREW 1.0 .................. 20
4.1 Simplified Class Diagram showing the Interaction of the UI and the Program Logic ........................................... 26
4.2 Preference Dialog ......................................................... 28
4.3 Example: default vs. increased font size ................................. 29
List of Listings

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creating a new Keystore</td>
</tr>
<tr>
<td>2</td>
<td>JNLP Security Settings</td>
</tr>
<tr>
<td>3</td>
<td>Signing a JAR File</td>
</tr>
<tr>
<td>4</td>
<td>Verification of the Digital Signatures</td>
</tr>
<tr>
<td>5</td>
<td>An Example of Program Logic in the GUI</td>
</tr>
<tr>
<td>6</td>
<td>Test Case for RuleML: Study Course with &lt;Implies&gt;</td>
</tr>
<tr>
<td>7</td>
<td>Test Case for RuleML: Empty Relation and Individual</td>
</tr>
<tr>
<td>8</td>
<td>Test Case for RuleML: Omitted Rulebase Tag</td>
</tr>
<tr>
<td>9</td>
<td>Test Case for RuleML: Get Address</td>
</tr>
<tr>
<td>10</td>
<td>Example of invalid RuleML 0.91 accepted by OOjDREW 0.961</td>
</tr>
</tbody>
</table>
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTLR</td>
<td>Another Tool for Language Recognition</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>DI</td>
<td>Dependency Injection</td>
</tr>
<tr>
<td>DRY</td>
<td>Don't Repeat Yourself</td>
</tr>
<tr>
<td>DVCS</td>
<td>Distributed Version Control System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive</td>
</tr>
<tr>
<td>jDREW</td>
<td>Java Deductive Reasoning Engine for the Web</td>
</tr>
<tr>
<td>JNLP</td>
<td>Java Network Launching Protocol</td>
</tr>
<tr>
<td>KB</td>
<td>Knowledge Base</td>
</tr>
<tr>
<td>LGPL</td>
<td>Lesser General Public License</td>
</tr>
<tr>
<td>LOC</td>
<td>Lines Of Code</td>
</tr>
<tr>
<td>OO jDREW</td>
<td>Object Oriented jDREW</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>POSL</td>
<td>Positional-Slotted Language</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOC</td>
<td>Separation of Concerns</td>
</tr>
<tr>
<td>SRP</td>
<td>Single Responsibility Principle</td>
</tr>
<tr>
<td>SVN</td>
<td>Subversion</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>VCS</td>
<td>Version Control System</td>
</tr>
<tr>
<td>QA</td>
<td>Quality Assurance</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XOM</td>
<td>XML Object Model</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

OO jDREW is an object oriented extension of the Java Deductive Reasoning Engine for the Web (jDREW). The application serves as a reference implementation of the (NAF Hornlog) RuleML web rule language and provides object oriented extensions for order sorted types, slots and object identifiers [1].

jDREW is a state–of–the–art deductive reasoning engine used for querying and processing facts and rules by using Prolog, Positional-Slotted Language (POSL), or RuleML syntax. Both, jDREW and Object Oriented jDREW (OO jDREW), are written in, and for use with, the Java language. Furthermore, (OO) jDREW provides two major execution approaches: a top–down (backward reasoning) mode for extracting information from Knowledge Bases (KBs) using queries, and a bottom–up (forward reasoning) mode to deduce knowledge from a specified set of facts and rules.

However, according to the OO jDREW Wiki [2] and to experimental tests executed by the current authors, the software lacks usability and a number of features. Revising OO jDREW in this way would make the application easier to use, enhance its web integration and potentially permit more individuals and businesses to make use of it.

Therefore, the authors decided to focus on the improvement of OO jDREW. For this reason, it is necessary to integrate support for the current RuleML 1.0 specification [3] [4] and furthermore, to improve its usability as much as possible within the given project time.
1.1 Issues

In addition to the problems listed in the RuleML Wiki \([5]\), we have identified the following missing features and problems, which come with \([\text{OO jDREW}]\) in version 0.961:

- **RuleML support**
  - Parser does not accept valid RuleML 0.91 documents\(^1\)
  - Parser does accept invalid RuleML (for example see listing\(^10\))
  - System outputs RuleML which does not validate against the RuleML schema specification\(^2\)
  - No support for RuleML 1.0

- **Project management**
  - No public source code control
  - No issue tracking
  - Outdated project roadmap

- **Documentation**
  - Current JavaDoc (source code documentation) effort is insufficient
  - Little high-level documentation for the internals of \([\text{OO jDREW}]\) (data structures, reasoner, etc.)

- **Overarching issues**
  - Code looks rather unmaintainable and requires major refactoring
  - Dead code / useless dependencies (e.g. mysql-connector-java-5.0.4.jar)
  - Duplicated code all over the place
  - Questionable coding style\(^3\)(use of untyped generics, magic constants, string-typing, 1000+ Lines Of Code (LOC) methods, etc.)

---

\(^1\)for example: \[\text{http://ruleml.org/0.91/exa/datalog/discount.ruleml}\]

\(^2\)no well-formed Extensible Markup Language (XML), does not conform to \[\text{http://ruleml.org/0.91/xsd/}\]

\(^3\)cf. [6]
Objectives

- Testing & Quality Assurance (QA)
  - Test infrastructure does not exist
  - Unit testing is impossible (due to current software design)

- User interface
  - User Interface (UI) mixed with program logic
  - Copy & paste issues (except on X11)
  - Cannot use look & feel provided by the underlying operating system

1.2 Objectives

It is obvious, that we cannot solve every issue identified in section 1.1 during our project work. Therefore we confine ourselves to a limited number of problems:

- Support for RuleML 1.0
  - Identify changes from RuleML 1.0
  - Rewrite parser infrastructure to support RuleML 1.0
  - Differentiate between RuleML sub-languages

- Project infrastructure
  - Move source code repository from SourceForge to GitHub
  - Set up bug tracking and a proper project roadmap

- Refactoring & Other features
  - Source code cleanup
  - Graphical User Interface (GUI) improvements
  - Definition of test cases

- Document and test the changes

In addition to the above mentioned objectives, further goals have been identified during the project work, particularly in terms of the RuleML parser (see section 3.3).
1.3 Approach

According to the aspects mentioned above, the authors have developed the following methodology in order to improve [OO jDREW] and make it “more attractive to the masses”:

Analysis & Design  In the first step, it is necessary to understand and document the differences from the earlier version of RuleML and to get an overall understanding of [OO jDREW]’s implementation, particularly of the RuleML parser. In parallel, we will check the RuleML 0.91 and RuleML 0.88 parser infrastructure for further problems. In order to meet our objectives regarding the RuleML parser, we will re-engineering and redesign the parser infrastructure. The source code analysis is described in section 3.1.

Most of the changes in RuleML only affect some XML tags, e.g. the role tags <head> and <body> in version 0.91 are replaced with <then> and <if> in the current version 1.0. Moreover, there are slight changes that were made at the attribute level, e.g. the attribute “in”=“no | semi | yes | effect | modal” has been replaced by “per”=“copy | open | value | effect | modal”. Section 3.2 provides more detailed information of the changes between RuleML 0.91 and RuleML 1.0.

In addition, we will check OO jDREW’s current version for possible problems related to the overall program and the user interface. Given this information, we will derive further improvements and objectives.

Finally, we will create test cases and enable automated parser testing by using the JUnit integration provided by the Maven build system (see section 2.4).

Implementation & Documentation  After the analysis, we will focus on the actual implementation. This work will be split up into two sub-teams, each of which will work on different subsets of the objectives specified in section 1.2. The first sub-team focused on the implementation of the new parser infrastructure and the new top-down user interface. This comprises the implementation of RuleML 1.0 support and the development of a new top-down user interface, while the other sub-team will define test cases, focus on problems like the copy & paste issues of OO jDREW’s Java Web Start version and look for a possibility to change the font sizes in the application. Of course, this also includes proper documentation for the source code, the test cases, results, possible problems and future work.
1.4 Tools

Our project work was based on the source code of OO jDREW 0.961, which is available through SourceForge⁴.

For the source code editing and the new user interface we used the Eclipse Integrated Development Environment (IDE)⁵ with the Java development tools (JDT) and Google's WindowBuilder⁶.

For the version control of the source code, we made a transition from the central version control system Subversion to the Distributed Version Control System (DVCS) Git. This transition is further described in section 2.1.

⁴see http://sourceforge.net/projects/oojdrew/
⁵see http://eclipse.org/
⁶see http://www.eclipse.org/windowbuilder/
Chapter 2

General & Miscellaneous Improvements

Besides the improvements and changes described in chapters 3 and 4, the authors made several changes related to the project infrastructure to achieve the goals described in section 1.2.

2.1 Transition to GitHub

Prior to this project, the source code of OO jDREW was maintained in a private Subversion (SVN) repository and stable versions released in SourceForge, despite the fact that OO jDREW is both an open source project and the reference implementation for RuleML. Therefore, the project should be open to the entire semantic web community at all times.

The authors decided to use Git instead of SVN as Version Control System (VCS) for a variety of reasons. First, Git belongs to the family of the so-called DVCS which are the state-of-the-art solution for version control. DVCS give developers a lot more flexibility than traditional client-server systems by eliminating the need for a central server and the notion of a “blessed” master repository. Since every developer has his own clone of the project, there is not even a need for elaborate permission management or backups, which are typical examples for unpopular tasks among developers.

Of course, there are other DVCS as well, but Git seems to be the most prevalent of them and happens to be the one that the authors are already familiar with.

1see http://git-scm.com/

2Git uses the term “clone” rather than “copy” to reflect the fact that it deals with full-fledged, equal repositories containing the project’s complete history
Git itself brings all tools which are necessary to import an SVN repository. During the import operation, Git replaces the domain part of the committers’ eMail addresses by randomly generated hex-strings. It is possible to keep the full address, but we have chosen not to do that in order to protect the privacy of the original contributors.

After the import, the authors created an organization called “OOjDREW” on GitHub, a platform that provides excellent Git integration and free hosting for open source projects.

2.2 Reorganization of the Project Structure

The next step we performed after we had the project’s history available in Git, was to reorganize the file structure. Originally, the repository contained many files that were obviously unrelated to OOjDREW. Those files included:

- Java Archive (JAR) files (external dependencies)
- Java .class files
- Java source files
- miscellaneous files

Most of the JAR files were removed since dependency management is the responsibility of the build system (see section 2.4). The Java class files are the result of compiling the Java source files and it is generally considered a poor practice to store generated or intermediate files in a repository. They are not relevant for the project history and keeping them is just a waste of space. Finally, a number of Java source files seemed to be demonstrations and showcases that have been developed for conferences and therefore are not relevant for the project itself. A complete list of all removed files is given in section A.6.

After the removal of all unnecessary files, the source code was restructured according to common Java coding conventions. The top-level Java package is now called “org.ruleml.oojdrew” instead of just “oojdrew.oojdrew” and some classes were moved to different packages (e.g. the files of the POSL parser generated by Another Tool for Language Recognition (ANTLR) now reside in their own

---

3see https://github.com/organizations/0OjDREW] Note that GitHub does not allow names to contain whitespaces.
Re-Engineering Effort

Figure 2.1: Packet Structure of OO jDREW

package, namely “org.ruleml.oojdrew.parsing.generated”). Figure 2.1 shows the current package diagram in Unified Modeling Language (UML) notation. Notice that there are some circular dependencies between the packages, namely between “GUI” and “TopDown” as well as between “parsing” and “util”. However, this is not a big issue and cannot be avoided (the association between the top-down user interface and its program logic has to be bidirectional).

2.3 Re-Engineering Effort

Besides the changes related to the user interface and the RuleML parser (see sections 3 and 4), the authors applied many smaller changes to improve the overall code quality. Ambiguous identifiers have been renamed wherever possible to enhance the readability of the code. One early example for such an improvement would be the
class SubException which has been renamed to SubsumesException since “Sub” is normally a prefix that can be used in a huge number of contexts (e.g. subclasses, see 6 chapter 2).

Another poor coding practice we observed was to define multiple classes in a single Java source file. The most prominent example for this issue are the classes POSLTypeQueryException and RuleMLTypeQueryException which were defined in a single source file. This is only allowed for inner classes and Java demands, that each class is defined in its own file where the file name reflects the class name.

Most source files of OO jDREW contained a huge amount of import statements. While those statements have no impact on the code size or the performance, they make it hard to understand the true dependencies between the individual modules (six, p. two, seven). It is easy to resolve this matter by using Eclipse, which is able to remove unused imports.

While reading and trying to understand the source code of OO jDREW, the authors discovered that a large portion of the code was duplicated several times (cf. Maven 'site' goal, section 2.4), which is a violation of the Don't Repeat Yourself (DRY) principle ([6 p. 48]). The most obvious case of code duplication was the entire RuleML parser, which was effectively included three times (for RuleML 0.88, RuleML 0.91 and RuleML queries, see chapter 3 for details). Also, we found many smaller identical (or almost identical) code blocks in different places. We removed the duplicates as soon as we encountered them, but since we did not tackle the entire source code, there are still duplicates left.

When we first imported the source code into Eclipse, we noticed that OO jDREW uses Java's generic types without parameters (e.g. Vector instead of Vector<String>). While this is not forbidden, it prevents the Java compiler from doing type checks at compile time, which can lead to subtle, hardly debuggable runtime errors ([6 p. 115]). We fixed only a few instances of this problem. To add a parameter to a generic type, one has to understand which types are used or stored in the object. This means that it would be necessary to read and understand the entire source code to solve this issue once and for all.

A more cosmetic change that was applied to all source files was the license text at the beginning of each file. Previously, licensing was not applied consistently and some files had just a comment that contained information about which software the file belongs to. We added the short Lesser General Public License (LGPL) header to all source files, crediting the two original authors. Furthermore, a LICENSE file

---

Re-Engineering Effort

containing a full copy of the LGPL was added to the top-level directory.

After we applied the changes mentioned in section 2.4.1 we noticed that trying to parse a RuleML document gave us an error message saying "Content is not allowed in Prolog". Further investigation showed, that some parts of the source code contained hard-coded offsets for string-processing operations, like:

```java
String s1 = d.substring(23);
```

We replaced those hard-coded offsets with code that is able to determine the correct values at run-time.

The transition to Maven (see section 2.4) allowed us to take some quality metrics of the source code. This analysis showed that there were a lot of code snippets like the following one:

```java
int i = new Integer(23);
```

This is not optimal since the construction of an actual integer object is quite expensive in terms of processing time and memory consumption. Even worse, the new object will be completely useless since it is instantly converted to a primitive type. Since the reasoning engine uses integers for its internal representation of symbols, this could contribute to the poor performance of OO jDREW when working with large knowledge bases. We removed all instances of this problem, but did not take any performance measurements.

Also, the authors are not fond of the abundant use of the static keyword in the source code. There are many classes which expose their internals by using public and static attributes. These attributes are effectively global variables that can be accessed and modified from anywhere. This clearly violates encapsulation and information hiding, which are principles advocated by the Java community ([6, p. 296]). We were able to remove some, but not all of the static definitions. A complete solution would require a careful redesign of some components.

Finally, we removed all code which was evidently unnecessary. This includes dead code (statements which will not execute under any circumstances), unused variables and a couple of other things ([6, pp. 288, 292]), which have been manually as well as automatically detected by the Maven 'site' goal (see section 2.4).

---

5 See [http://www.gnu.org/licenses/lgpl.html](http://www.gnu.org/licenses/lgpl.html)

2.4 Transition to Maven

Ever since the authors attacked the overhaul of OO jDREW, they were bothered by the necessity to take care of external dependencies like log4j. Those dependencies made running OO jDREW unnecessarily complicated. Starting the software without Eclipse or Java Web Start required the specification of a non-trivial shell command. Otherwise, Java would not be able to find all required JAR files.

To solve this issue, we decided to use Maven as build system. Maven is highly configurable and comes with a long list of nifty plug-ins. One of these plug-ins is even able to handle JAR signing (see section 2.6) automatically. Currently, the following goals are supported:

- **compile** Compiles all Java source files into .class files.
- **test** Executes unit tests using JUnit.
- **package** Creates two JAR files. Both contain all .class files generated during the compile phase. However, the second one also contains all external dependencies, which allows users to start OO jDREW with a simple double-click. If a key store is defined in pom.xml, the JAR files will be digitally signed.
- **site** Generates a Maven site which contains code quality reports obtained from a variety of tools. This allows developers to check for problems like duplicated code or methods with high cyclomatic complexity.

Especially the package goal, which produces a single JAR file, has proven itself very useful throughout this project. During initialization, Maven will fetch all external dependencies from a Maven repository on the internet. This allows us to keep the Git repository clean of external dependencies. However, there is one exception. OO jDREW uses the Directed Acyclic Graph (DAG) classes of the Ptolmey project for the type system. To the best of our knowledge, Ptolmey is not available on any public Maven repository which is why we have created a local fake repository containing the Ptolmey JAR file. This fake repository can be found in the “repo” folder of the Git repository.

---

7 see [http://maven.apache.org/](http://maven.apache.org/)
8 see [http://www.junit.org/](http://www.junit.org/)
10 see [http://ptolemy.eecs.berkeley.edu/](http://ptolemy.eecs.berkeley.edu/)
2.4.1 Removal of the custom XML Serialization Routines

OO jDREW uses the XML Object Model (XOM) library for XML processing. The original authors of OO jDREW modified the serialization routines of that library in such a way that the XML declaration

```xml
<?xml version="1.0" encoding="UTF-8" ?>
```

at the beginning of a document is omitted. We modified OO jDREW to work without these custom routines since they make it impossible to use the XOM library, which is automatically downloaded by Maven.

2.5 POSL Parser Update

One of the last changes was to include the updated POSL parser from Team. The integration went smoothly and working with the other team was a valuable experience.

2.6 JAR Signing

In OO jDREW 0.961 it was not possible to paste text from, or copy text to a web browser or any other document. The source of this problem were the security policies for Java Web Start applications. Those applications are not allowed to access critical system resources such as the clipboard or the file system.

Basically there are two ways to give an untrusted application access to the clipboard:

1. Modify the machine's local policy file
2. Digitally sign the JAR file of the application

The main difference between these two approaches is portability. While the policy file has to be modified for every machine, the digital signature has to be applied only once, hence was chosen here. The following paragraphs outline the necessary steps to create a so-called "keystore" and to modify the Java Network Launching Protocol (JNLP) file.

---


First, we need a [JAR] and a [JNLP] file. A [JAR] file is a zip archive that bundles all the resources of the application. A [JNLP] file contains the required instructions for Java Web Start.

**Listing 1 Creating a new Keystore**

```bash
X:\00jDREW>keytool -genkey -keystore keyStore -alias 00jDREW
Enter keystore password: ******
Re-enter new password: ******
What is your first and last name?
[Unknown]: John Doe
What is the name of your organizational unit?
[Unknown]: CS
What is the name of your organization?
[Unknown]: UNB
What is the name of your City or Locality?
[Unknown]: Fredericton
What is the name of your State or Province?
[Unknown]: New Brunswick
What is the two-letter country code for this unit?
[Unknown]: CA
Is CN=John Doe, OU=CS, O=UNB, L=Fredericton, ST=New Brunswick, C=CA correct?
[no]: yes
```

Enter key password for <00jDREW>
(RETURN if same as keystore password):

---

Signing [JAR] files works with public key encryption where two different keys are used: a private and public key. The private key is used to sign the [JAR] file, and must be kept secret. The corresponding public key can be used to verify the signature. Java uses a signature scheme which is based on X.509[13].

The tools to create a keystore and to sign applications are included in the Java Software Development Kit (SDK). Listing 1 shows how to use the “keytool” executable to create a personal keystore. Once this step is finished, the “jarsigner” tool can be used to create a digital signature for any [JAR] file (see listing 3). Afterwards, the digital signature can be verified by using the commands shown in listing 4. The last step is to specify the permissions of the application in the [JNLP] file. This can be accomplished by adding the XML fragment shown in listing 2 to the [JNLP] file.

JAR Signing

Listing 2 JNLP Security Settings

```
1: <security>
2:   <all-permissions/>
3: </security>
```

Listing 3 Signing a JAR File

```
X:\O0jDREW>jarsigner -keystore keyStore oojdrew-1.0.jar O0jDREW

Enter Passphrase for keystore: ******

Warning:
The signer certificate will expire within six months.
```
Listing 4 Verification of the Digital Signatures

X:\00jDREW>jarsigner -keystore keyStore -verify -certs -verbose oojdrew-1.0.jar

X.509, CN=John Doe, OU=CS, O=UNB, L=Fredericton, ST=New Brunswick, C=CA (oojdrew)
[certificate will expire on 3/14/12 3:16 PM]

smk  21046 Thu Dec 15 14:19:42 AST 2011 org/ruleml/oojdrew/util/Term.class

X.509, CN=John Doe, OU=CS, O=UNB, L=Fredericton, ST=New Brunswick, C=CA (oojdrew)
[certificate will expire on 3/14/12 3:16 PM]

smk  7863 Thu Dec 15 14:19:44 AST 2011 org/ruleml/oojdrew/util/Types.class

X.509, CN=John Doe, OU=CS, O=UNB, L=Fredericton, ST=New Brunswick, C=CA (oojdrew)
[certificate will expire on 3/14/12 3:16 PM]

s = signature was verified
m = entry is listed in manifest
k = at least one certificate was found in keystore
i = at least one certificate was found in identity scope

jar verified.

Warning:
This jar contains entries whose signer certificate will expire within six month.
Chapter 3

RuleML Parser

Extending OO jDREW by supporting RuleML 1.0 is one of the main objectives of the present project work.

The following sections describe the way in which the authors have extended OO jDREW to support RuleML 1.0 according to the specification [3]. Moreover, it is explained how the project team coped with the existing source code and the parser infrastructure given by OO jDREW 0.961 in order to improve the overall code structure and to implement the RuleML 1.0 knowledge base and query support.

3.1 Source Code Analysis

First of all, the project team analyzed the existing source code to gain an overall understanding of the program. This step has been done iteratively. Therefore, the developers first did some experimental tests with the program. For example, different kinds and versions of RuleML knowledge bases and queries have been used to produce different kinds of parsing errors and outputs. After that, the team, step by step, dived into the source code, or, more precisely, into the different RuleML parser classes that were already implemented (see figure 3.1).

During the parser analysis mentioned above, the team members came up with some problems related to the implementation. First of all, the RuleML 0.88+ and the RuleML 0.91 parser did not accept various kinds of valid RuleML in terms of the related specification (see [3], [7], [8]). For example, the parser threw an error, when the input document contained a <RuleML> tag or an XML header (i.e. <?xml version="1.0" encoding="UTF-8"?>). Moreover, the parser was not able to process some specific RuleML tags like the <oid> tag or the role tag <arg>. Furthermore,
the parser was only able to parse the "stripe-skipped" syntax defined by RuleML (0.88 and 0.91). In this kind of syntax role (edge) elements are omitted, relying on the canonical order of RuleML elements. For example, the application threw an error as soon as the knowledge base input contained striped syntax (see figure 3.2).

Another aspect, which the authors would like to mention at this point, is that the existing source code related to the RuleML parser has not been implemented in a cleanly structured and maintainable way. Also, the team has detected a lack of Java source code documentation. Only the most important functionality has been documented.

In the viewpoint of the authors, the implementation could have been done in a better way, so that future extensibility and better readability of the code could have been provided. There was only one RuleML parser class, namely RuleMLParser, which internally consisted of the RuleMLParser class itself and three inner classes. Furthermore, the inner classes (RuleML91Parser, RuleML88Parser and RuleMLQueryParser) contained a lot of duplicated code, like the tag name string constants or methods responsible for the document parsing itself. According to an estimation calculated by the project team, there was about 96 percent of code
Source Code Analysis

3.1.1 Internal Data Structures of the RuleML Parser

Knowledge of RuleML and XML parsing alone is not sufficient to write a new RuleML parser. The authors had to understand some of the mechanisms the parser used to translate the input document into the internal representation of the reasoning engine. In order to give new developers easy access to the parser, some of these mechanisms are documented below. We definitely recommend to take a look at the implementation of `Term` and `DefiniteClause` since these classes are the primary data structure used by the reasoner. However, the focus of this section is `OO jDREW`'s type system.

The first thing that we want to discuss is the `Types` class. This class implements `OO jDREW`'s type system and provides static definitions for all built-in types like strings or real numbers. This class is used by the parser to map the contents of the `type` attribute (which can be specified for a variety of RuleML XML elements) to a unique numeric identifier.

The class `SymbolTable` works in a similar fashion. It maps symbol names (e.g. from `<Var />`, `<Ind />`, `<Data />`, etc.) to unique numeric identifiers. In addition to that, it provides constants for some built-in symbols and roles like `<oid />`.

To parse skolem constants, the parser automatically generates symbolic names like `gensym id` based on the name of the skolem constant. Here, `id` denotes a serial number derived from the `SymbolTable`. To map the name of the skolem constant to the generated symbolic name, a hash table called `skolemMap` is used.

Something that the authors do not fully understand yet, is the purpose of the parser's `variableNames` field. This vector is used to store symbol names generated by the `internVariable` method which is used to create fake `<oid />` tags for `<Atom />` elements and to generate numeric identifiers for `<Var />` elements. In theory, it should be possible to use the `SymbolTable` for this purpose.

The last thing we want to discuss here is the `varClasses` field. This hash table stores a list of types assigned to each symbol, the default being "thing" (`IOBJECT`). Whenever the parser encounters a new fact or implication, it uses this table to derive a single type for each symbol. To illustrate this behavior, let's assume that there is a variable "v0" that is used with both the types "Car" and "Ship" in a fact or rule. In that case, the parser would call the `buildTypeTable` method to replace these types by their greatest lower bound class (e.g. "Amphibious"). After this step the method...
Derived Objectives must be used to update the type of the variable in every term created so far.

3.2 Difference between RuleML 1.0 and RuleML 0.91

In this step, the authors took a look at the difference between RuleML 1.0 and RuleML 0.91, particularly at the ones which affected the RuleML parser in OO jDREW. After doing some comparison of the corresponding specifications, the authors came up with the following differences:

The most important change between the two RuleML versions, which also affected OO jDREW’s RuleML parser, was the modification of the premise and conclusion tags. `<head>` and `<body>`, `<lhs>` and `<rhs>` in version 0.91 of RuleML were replaced by `<then>` and `<if>`, `<left>` and `<right>` in version 1.0. Moreover, RuleML 1.0 introduced a new child tag for the `<RuleML>` root tag, namely the `<act>` tag. These above mentioned changes are the most important ones and the ones the author focused on while implementing the support for parsing RuleML 1.0. However, some additional changes were made related to some attribute and type tags. But since OO jDREW’s internal reasoning engine is not yet able to process this kind of syntax, the authors decided to skip them.

3.3 Derived Objectives

According to the aspects mentioned above, and to improve and extend OO jDREW’s RuleML parser infrastructure, the project team came up with the following objectives:

- Remove duplicated code
- Merge RuleML parser classes
- Add strict typing to generics (e.g. “Vector<DefiniteClause>”)
- Create one consistent parser class
- Outsource hard-coded variables to a separate class (e.g. tag names)
- Implement support for striped as well as stripe–skipped syntax
- Extend source code documentation
Technical Description

- Maintain backward compatibility to RuleML 0.88, 0.89 and 0.91
- Provide an optional XML validation against specified XSD

3.4 Implementation Methodology

During all these steps, the authors strictly aimed at following the Single Responsibility Principle (SRP) \([9]\). Therefore, each object, class and method should be modified to only have a single, well-defined responsibility. The implementation of these features has been done iteratively. By using this methodology, the authors were able to understand, enhance and clean-up the code step by step.

3.5 Technical Description

The new parser infrastructure is depicted in figure 3.3

Figure 3.3: New RuleML parser infrastructure in OO jDREW 1.0

The figure visualizes the new parser infrastructure, in which the original three parser classes have been aggregated into one core parser class. This class is responsible for parsing the contents of a given RuleML document and it is able to cope with the different versions of RuleML (from 0.88 to 1.0).
RuleMLTagNames  The RuleMLTagNames class now contains all RuleML element names, which are required to parse a RuleML document. Furthermore, our project team extended the tag names in order to add basic support for NAF Hornlog and more expressive RuleML 1.0 sublanguages. For example, we added tag names like `<if>`, `<then>`, `<Reify>`, `<formula>`, `<torso>` and `<degree>`.

RuleMLDocumentParser  As already stated, this parser class is a replacement for the original three parser classes. It also contains improvements related to source code readability and maintainability. We introduced various new methods like a method which is responsible to parse the RuleML root elements and omit optional elements like `<RuleML>` or `<Rulebase>`. In order to get rid of errors when parsing documents containing elements which are not supported by `OO jDREW`'s internal reasoning engine yet, we provided methods (e.g. `skipRoleTag()` or `removeReifyElements()`) for skipping such elements. Another new feature of the parser is the ability to parse and query empty relations (`<Rel />`) and empty individuals (`<Ind />`).

RuleMLParser  This class provides an additional abstraction layer above the RuleML document parser. It is used to invoke the document parser for different kinds of inputs, like files, file paths, or by directly inputting a RuleML document stored in a string. Finally, this parser class can be configured to validate the input document against its XML Schema (XSD). This option can easily be configured via the preference dialog (see section 4.2).

3.5.1 Implementation Issues

During the realization of the aforementioned objectives, the team had to cope with some difficulties. First of all, it was necessary to study the specifications carefully, because the RuleML 1.0 specification does not provide an explicit description of the attribute mappings (e.g. `mapClosure`). Another aspect is the complexity of parsing RuleML. For example, context-sensitive role tags, which can have different semantics at different locations, made the implementation slightly more difficult. Moreover, it was necessary to get an overall understanding of the internal data structures used by `OO jDREW` and how the parser interacts and exchanges data with the forward and backward reasoning engine. Finally, one remaining issue which the project team was not able to solve yet, is to disconnect the dependency between the POSL and the RuleML parser.
3.6 Test Cases

In this section, we will provide some test cases with the purpose of testing the changes applied to OO jDREW. Moreover, the test cases should prove that the modified OO jDREW is still working as intended.

3.6.1 Premise and Conclusion

This first test case has the purpose to test the RuleML 1.0 Implies element with a premise and a conclusion tag (<if> and <then>). These elements were introduced in the new OO jDREW. The corresponding RuleML 1.0 test case document can be found at appendix A. The document does also contain a corresponding query and the expected result(s).

3.6.2 Empty Relation and Individual

This second test case aims at the verification of the parser when using a knowledge base containing an empty relation and an empty individual (see appendix A.2).

3.6.3 Omitting the Rulebase Element

This is a simple test case which is used to check if the parser still works correctly, even when the <Rulebase> element is omitted (see appendix A.3).

3.6.4 Get Address with Equation Element

For this test case it is important to notice that some tags are not supported by OO jDREW’s internal reasoning engine yet. An example for that is the <Equal> element. However, the RuleML parser still accepts it (see section 3.5). The corresponding RuleML test case can be found at appendix A.4. The test case represents an example about getting information for an address. In contrast to the previous test cases, in this case the query will not provide all results that are given by the knowledge base. It will fail when a query about <Ind>New Brunswick</Ind> is executed even though <Ind>New Brunswick</Ind> should be equal to <Ind>NB</Ind>. Other queries, which do not make use of the equals definition, will work seamlessly.
3.7 Results

The results of this part of the project is the realization of the objectives explained in section 3.3. The outcome is very obvious. The team was able to deal with every predefined goal. The LOC related to the parser infrastructure have been reduced by approximately 60 percent from 3961 to 1433. Moreover, the new infrastructure enables the application to parse valid RuleML in either striped or stripe-skipped syntax, and it also accepts all RuleML 1.0 role tags where applicable. Another nice feature of the new infrastructure is the optional XML validation. This feature can be enabled via the user interface and validates the input document against its XSDs, if network connectivity is available.
Chapter 4

User Interface

As mentioned earlier, the authors were not satisfied with the UI of OOjDREW 0.961. This dissatisfaction included both the functionality and the implementation of the UI.

From a software engineering point of view, the most serious design error of the old UI was to mix program logic with other things like GUI code or object graph construction. Listing 5 illustrates this issue. This snippet of code was taken from the old top-down GUI and is part of an action listener. Naturally, there is absolutely no sensible reason to do XML processing in the UI. This processing was thus moved to the corresponding application classes (TopDownApp/BottomUpApp).

Listing 5 An Example of Program Logic in the GUI

```java
1: if (root.getFirstChildElement("And") != null) {
2:     and = root.getFirstChildElement("And");
3:     Elements children = and.getChildElements();
4:     atom2 = new Element(and);
5: }
6: else {
7:     Element atom = root.getFirstChildElement("Atom");
8:     atom2 = new Element(atom);
9: }
```

Other shortcomings of the old UI included:

- Fixed UI font size
- Fixed window size
Technical Description

- Malfunctioning copy & paste
- Outdated file I/O dialogs (load/save file)
- No way to load type definitions or queries from a file / web source
- No way to save type definitions, knowledge bases or queries

After a short evaluation, the authors decided to rewrite the GUI from scratch. This was also a good opportunity to apply some well-known best practices like the SRP (Separation of Concerns) and Dependency Injection (DI). The scaffolding for the new GUI was created using WindowBuilder, a GUI designer for Java that has been released by Google as an open source product. Some of the features of the new GUI are described in section 4.2.

4.1 Technical Description

Figure 4.1 shows a simplified class diagram that illustrates the interaction between the different components of the new top-down GUI.

The class TopDownApp plays an important role since it implements the entire program logic of the application. In order to instantiate this class, a static factory method must be used. This factory method is responsible for the construction of the object graph and wires all the different pieces of the application together.

This approach simplifies testing because it is possible to replace specific parts of the application by mock-ups for testing purposes. Let us take a look at the RuleML parser. The parser needs a Configuration object to find out which RuleML version the user selected and whether it should validate the RuleML document or not.

Instead of constructing the configuration object itself, the RuleML parser explicitly asks for this object in its constructor. This allows developers to provide the parser with a fake configuration for unit tests. Otherwise, the parse would always use a real configuration object and running any unit test would possibly result in system-wide configuration changes.

If we apply this technique to a less abstract real world concept, its benefits become clear. Imagine that we have a Car class that needs an Engine object. If we had a line of code like

```java
this.engine = new Engine()
```

4.1 see [http://www.eclipse.org/windowbuilder/](http://www.eclipse.org/windowbuilder/)
Figure 4.1: Simplified Class Diagram showing the Interaction of the UI and the Program Logic
in the constructor of Car, then it would be impossible to test the car in isolation (without the engine). In the real world, this situation would be rather bizarre because a car does not know how to build an engine. Instead, there is a car factory that builds the individual parts and assembles them.

This is an application of the SOC and DI principles which allow developers to instantiate tiny fractions of their applications and test them in isolation. This was an impossible thing to do with OO jDREW because of entanglement and many cross-class dependencies. All components shown in figure 4.1 have been developed with respect to SOC and DI.

Preference and Configuration Management

Another part of the new application design, that requires some explanation is the preference system. When the user closes the preference dialog using the “Ok” button, the preference dialog sends a message to the controller (TopDownApp) that requests an update of the preferences. The controller then reads the settings from the preference dialog and applies them to the configuration object. The configuration is based on the Java preference Application Programming Interface (API)\(^2\) and fires an event whenever a setting changes.

This event is processed by a number of event handlers. For example, the event handler in the PreferenceManager class will update the font size and look of the UI components when these settings change. However, we noticed that changing the font size or look and feel at run time will result in an exception from an unknown source. This exception is caught by the UI’s event queue and does not seem to affect the rest of the application. Still, this could indicate a problem and should be subject to further investigation.

4.1.1 Custom UI Components

To implement all features of the new UI, it was necessary to write a couple of custom UI components. The first custom UI is the query tab. It uses two split panes to separate the query from the result horizontally and one to separate the proof tree from the table containing the variable bindings. When the window is resized, the latter split pane should distribute the available space equally between the proof tree and the binding table. Unfortunately, this behavior cannot be achieved with standard

\(^2\)see \url{http://docs.oracle.com/javase/1.4.2/docs/guide/lang/preferences.html}
Swing components. We had to implement a custom component that inherits from `JSplitPane` and modify its `paint` method to get the desired result.

Surprisingly, severe bugs seem to exist in Swing. The distribution of space between the left and the right half depends on the speed with which the window is being resized. During the development of the new UI we discovered several other idiosyncrasies like this. This suggests that Swing is no longer contemporary and it might be prudent to switch to a more up-to-date UI tool kit.

The second custom UI component was needed in order to implement undo and redo functionality in `OO jDREW`. The text areas supplied by Swing do not implement this ubiquitous feature and burdens developers with implementing their own solution to this problem. Therefore, we implemented a custom text area control that provides undo/redo functions and a context menu.

### 4.2 Preference Dialog

Due to further usability lacks in `OO jDREW`, the authors decided to make further improvements to the top-down user interface. Therefore, a preference dialog (see figure 4.2) has been introduced by the authors of the present work. The preference dialog can easily be accessed via the menu bar or via the keyboard shortcut (CTRL + P). The main purpose of this dialog is to provide users the possibility to easily modify the UI settings and change them according to the users' needs. Moreover, the dialog is used to change the RuleML version according the input format. Therefore, the following functionality has been implemented during the project work:

![Figure 4.2: Preference Dialog](image)
Font size setting  One major improvement which is particularly useful for presentations, is the opportunity to adjust the font sizes. This adjustment can be done in two ways. The first way allows the user to modify the font size of the text areas and the main user interface separately ("Link font sizes" checkbox unchecked). The second way is to modify the font sizes over the whole application by only changing one spinner. This can be done by checking the "Link font sizes" checkbox (see figure 4.2).

On the first startup of the application, the font sizes are initialized with the default value 12. However, each time, when the settings are changed and the changes are applied, it will be recognized and saved by the internal configuration manager (cf. section 4.1).

Look and Feel  Another nice feature is provided in the new **OO J Drew** the ability to change the look and feel, or more precisely, the overall appearance of the program. The options which can be selected at this point can differ from one operating system to another. For example, the Windows and Windows Classic style will not be available in Linux, because they directly depend on the API of the underlying operating system.

RuleML version  Finally, the last combobox allows to select a specific RuleML version. If this setting is changed, it will be passed to the RuleML parser. This option is also important because it tells the RuleML parser which version it has to parse and which tag names it has to load (see chapter 3 for details).
Chapter 5

Conclusion

5.1 Outlook & Future Work

Throughout the project, the authors identified a number of previously unknown issues and came up with many ideas to improve OO jDREW further.

Our experience with the project showed that some parts of the software definitely require more refactoring or even a complete redesign. The abundant use of static methods and variables as well as a lack of encapsulation will be a major obstacle should OO jDREW ever be extended to take advantage of the full power of modern processors. There is also a strong need for a comprehensive test suite that not only tries to parse knowledge bases, but also issues queries and verifies their results.

The next logical step after that would be to incorporate our enhancements into the other parts of OO jDREW, namely the bottom-up application and POSL-RuleML converter \[ \text{[10]} \]. This has already been started as part of the GitHub project\[ \text{[1]} \].

Further feature enhancements would include the ability to validate RuleML documents against RelaxNG schemas and to extend the RuleML support to handle things like \(<\text{Equal} />\) or slotted rest elements\[ \text{[2]} \]. Developers would need deep insight into the reasoning engine and more time and more time. Unfortunately this was clearly beyond the scope of this project. One might even consider to rewrite the reasoner from scratch to make it more easily understandable – which is a desirable property for a piece of software that should serve as the reference implementation of RuleML.

Another nice-to-have feature would be to have a GitHub integration in Maven.

1\text{cf.} https://github.com/OOjDREW/OOjDREW
2https://github.com/OOjDREW/OOjDREW/issues/1
Currently, JAR files and Java documentation must be manually uploaded to GitHub. Automating these tasks would be a welcome relief. There is another unsolved issue regarding Maven. Ptolmey is currently the only external dependency that is not fully managed by Maven (see section 2.4 for details). This is not exactly a high-priority task, but still something worth the effort.

If we take a look at the future of OO jDREW, we can imagine some unique selling points such as parallel and distributed reasoning, partial knowledge base update, and syntax highlighting / autocompletion for RuleML and POSL.

To reach these goals, the project needs skilled and dedicated developers. Therefore, it might be worth to find new ways to promote the project and attract innovative developers and inquisitive researchers all over the world. It is an open source project after all, which is available through the GitHub social coding platform.

5.2 Summary

Working on the project with an international team has been a valuable experience for all participants, regardless of minor scheduling and coordination problems. We have achieved all of our goals and enhanced OO jDREW way beyond the original scope of the project.

We hope that our improvements will encourage people to play around with OO jDREW to learn more about reasoners for the semantic web. We also hope that our effort to make OO jDREW a real open source project will be the cornerstone of a software that can rejoice in an increasing popularity in the semantic web community.

\[3\text{see } \text{https://github.com/OOjDREW/OOjDREW/issues/4}\]
Bibliography


Appendix A

Listings

A.1 Test Case 1: Study Course

Listing 6: Test Case for RuleML: Study Course with <Implies>

```
1: <RuleML>
2:   <Assert mapClosure="universal">
3:     <Rulebase>
4:     <!--
5: This example rulebase contains four clauses.
6: The first and second clauses are implications;
7: the third and fourth ones are facts.
8: -->
9:     <Implies>
10:     <then>
11:       <Atom>
12:         <Rel>engage</Rel>
13:         <Var>student</Var>
14:         <Var>course</Var>
15:       </Atom>
16:     </then>
17:     <if>
18:       <!-- explicit 'And' -->
19:       <And>
```
26:
   <Atom>
27:       <Rel>take</Rel>
28:       <Var>student</Var>
29:       <Var>professor</Var>
30:       <Var>course</Var>
31:    </Atom>
32:
33:
34:
35:
36:
37:    </And>
38:    </if>
39:    </Implies>
40:
41: <!-- The second rule implies that student takes a course by 
42: a professor if a professor give a course for that student. -->
43:    <Implies>
44:    <then>
45:        <Atom>
46:            <Rel>take</Rel>
47:            <Var>student</Var>
48:            <Var>professor</Var>
49:            <Var>course</Var>
50:        </Atom>
51:    </then>
52:    </if>
53:    </Implies>
54:
55:
56:
57:
58:
59:
60:
61:
62: <!-- The third rule is a fact that asserts that Fred 
63: give CS6795 for John. -->
64: <Atom>
65:    <Rel>give</Rel>
66:    <Ind>Fred</Ind>
67:    <Ind>John</Ind>
68:    <Ind>CS6795</Ind>
69: </Atom>
70:
A.2  Test Case 2: Empty Relation and Individual

Listing 7: Test Case for RuleML: Empty Relation and Individual
Test Case 2: Empty Relation and Individual

1: <RuleML>
2:   <Assert mapClosure="universal">
3:     <Rulebase>
4:       <!--
5:           This example is showing the ability of parsing an empty
6:           relation with either an empty variable or an empty
7:           individual. Also, OO jDREW can come up with an empty
8:           result when query this example.
9:       -->
10:     <Atom>
11:       <Rel />
12:     </Atom>
13:   </Rulebase>
14: </Assert>
15:  <Query>
16:   <Atom>
17:     <op>
18:       <Rel></Rel>
19:     </op>
20:   <Var>Y</Var>
21: </Atom>
22: </Query>
23: <!--
24:   Specified Results
25:   Success
26:   Relation & Variables
27:   <Rel />
28:   <Var>X</Var>
29:   <Var>Y</Var>
30:   Value
31:   X = bound to X (OO jDREW: \textunderscore X)
32:   Y = anonym
33:   Actual Results
34:   Success
35:   -->
36: </RuleML>
A.3 Test Case 3: Omitted Rulebase Tag

Listing 8: Test Case for RuleML: Omitted Rulebase Tag

```xml
<RuleML>
  <Assert mapClosure="universal">
    <!-- This example is showing the ability of parsing an RuleML document in which the <RuleBase> tag has been skipped. -->
    <Atom>
      <Rel>f</Rel>
      <Var>X</Var>
      <Ind>Y</Ind>
    </Atom>
  </Assert>

  <Query>
    <!-- POSL f(\textit{X}, \textit{Y}) -->
    <Atom>
      <Rel>f</Rel>
      <Var>X</Var>
      <Var>Y</Var>
    </Atom>
  </Query>

  <!-- Specified Results Success -->

  Relation & Variables
  <Rel>f</Rel>
  <Var>X</Var>
  <Var>Y</Var>

  Value
  \textit{X} = \textit{bound to} \textit{X} (OO jDREW: X_0)
  \textit{Y} = \textit{Y}

  Actual Results
  Success
  -->
</RuleML>
```
A.4  Test Case 4: Get Address

Listing 9: Test Case for RuleML: Get Address

1:  <RuleML>
2:    <Assert mapClosure="universal">  
3:      <Rulebase>
4:        <!-- This example rulebase contains 3 clauses. The first is a relation of getting address information. -->
5:          <Atom>
6:            <Rel>getAdd</Rel>
7:            <Var>stAdd</Var>
8:            <Var>stNum</Var>
9:            <Var>posCode</Var>
10:           <Var>city</Var>
11:           <Var>province</Var>
12:         </Atom>
13:       </Rulebase>
14:    </Assert>
15:  </RuleML>
16:  
17:    <!-- The second is an equational formula consisting of two expressions. These two equations express that the individuals named "NB" and "New Brunswick" are the same, same as the individuals named "QC" and "Quebec" are the same -->
18:  <formula>
19:    <Equal>
20:      <left>
21:        <Ind>NB</Ind>
22:      </left>
23:      <right>
24:        <Ind>New Brunswick</Ind>
25:      </right>
26:    </Equal>
27:  </formula>
28:  
29:  <formula>
30:    <Equal>
31:      <left>
32:        <Ind>QC</Ind>
33:      </left>
34:      <right>
35:        <Ind>Quebec</Ind>
36:      </right>
37:    </Equal>
38:  </formula>
Test Case 4: Get Address

42:  </formula>
43:
44:  <!-- The third and fourth clauses are facts that assert
45:  information of an address. -->
46:
47:  <Atom>
48:      <Rel>getAdd</Rel>
49:      <Ind>Dunns Crossing</Ind>
50:      <Ind>20</Ind>
51:      <Ind>E3B 2A4</Ind>
52:      <Ind>Fredericton</Ind>
53:      <Ind>NB</Ind>
54:  </Atom>
55:
56:  <Atom>
57:      <Rel>getAdd</Rel>
58:      <Ind>rue Dalhousie</Ind>
59:      <Ind>85</Ind>
60:      <Ind>G1K 7A6</Ind>
61:      <Ind>Québec</Ind>
62:      <Ind>QC</Ind>
63:  </Atom>
64:
65:  </Rulebase>
66:
67:  <Query>
68:  <!--
70:  -->
71:  <Atom>
72:      <Rel>getAdd</Rel>
73:      <Var>Street</Var>
74:      <Var>StreeNo</Var>
75:      <Var>PosCode</Var>
76:      <Var>City</Var>
77:      <Var>Province</Var>
78:  </Atom>
79:
80:  <!--
81:  Specified Results
82:  Success
83:  Relation & Variables
84:  -->
85:  <Rel>getAdd</Rel>
86:  <Var>Street</Var>
Test Case 4: Get Address

Value [0]
Street = Dunns Crossing
StreetNo = 20
PosCode = E3B 2A4
City = Fredericton
Province = NB

Actual Results
Success

<Query>
<Atom>
<Rel>getAdd</Rel>
<Var>Street</Var>
<Var>StreeNo</Var>
<Var>PosCode</Var>
<Var>City</Var>
<Ind>New Brunswick</Ind>
</Atom>
</Query>

<!--
Specified Results
Success

Relation & Variables
<Rel>getAdd</Rel>
<Var>Street</Var>
<Var>StreeNo</Var>
<Var>PosCode</Var>
<Var>City</Var>

Actual Results
Failed
-->
A.5 Wiki — Roadmap update

== 00 jDREW 1.0 ==
* Bring 00 jDREW’s RuleML parser up to date for the newest RuleML 1.0 specification
  ** Redesign parser infrastructure
  ** Accept RuleML documents written according to the specification
  ** Differentiate between RuleML sub-languages implicitly
  ** Enhance parser infrastructure for RuleML 1.0 support

* Clean up, fix and enhance overall source code
  ** Remove duplicated and unused code
  ** Remove libraries which are referenced but not used
  ** Improve `\LaTeX` (copy/paste, undo/redo, font sizes)
  ** Enhance functionality, maintain source

* Software tests
  ** Create test cases and test report templates

* Update project infrastructure
  ** Switch to improved version control system (i.e. from SourceForge to GitHub)
  ** Add bug-tracking functionality (for future improvements and feedback)

A.6 Removed files

The following files were removed during the initial reorganization of the project. All paths are relative to the top-level directory.

- build.xml
- commons-codec-1.3.jar
- commons-logging-1.1.jar
- commons-logging-adapters-1.1.jar
- commons-logging-api-1.1.jar
- cos.jar
- doc/*
- index.html
• myfile40.txt
• mysql-connector-java-5.0.4-bin.jar
• resultsA.xml
• servlet-2_5-api.jar
• src/.classpath
• src/.project
• src/ChallengeChairRuleML2008.java
• src/DatabaseConnection.java
• src/DTRObject.java
• src/GeneralChairRuleML2008.java
• src/Globals.java
• src/HelloWorld.java
• src/HttpTest.java
• src/LiasonChairRuleML2008.java
• src/MessageGenerator.java
• src/MessageParser.java
• src/ModifiedTDGUI.java
• src/OOjDREWAPI.java
• src/OOjDREWAPITest.java
• src/OOjDREW3.java
• src/PanelChairRuleML2008.java
• src/PaperSubmittal.java
• src/ProgramChairAgent2.java
RuleML Listings

- src/PublicityChair.java
- src/PublicityChairRuleML2008.java
- src/QueryBuilder.java
- src/RouteAccumulation.java
- src/Server.java
- src/Servlet.java
- src/ServletTest.java
- src/SiteConnection.java
- src/ShowRequestHeaders.java
- src/TestConnection.java
- src/TopLevelOOjDREW.java
- src/VisitedNode.java
- src/VisitedObject.java
- src/WebChairRuleML2008.java
- src/siteExample.java
- src/programChair2.java
- taxonomy.rdfs

A.7 RuleML Listings

Listing 10: Example of invalid RuleML 0.91 accepted by OOjDREW 0.961
1: <Assert>
2: <Rulebase mapClosure="universal">  
3: <oid><Ind>Asserted content, i.e. rules and facts</Ind></oid>
4: 
5: <Implies>
6:  <And>
<Atom>
  <op><Rel>premium</Rel></op>
  <Var>customer</Var>
</Atom>

<Atom>
  <op><Rel>regular</Rel></op>
  <Var>product</Var>
</Atom>

<Atom>
  <op><Rel>discount</Rel></op>
  <Var>customer</Var>
  <Var>product</Var>
  <Ind>5.0 percent</Ind>
</Atom>

<Atom>
  <op><Rel>premium</Rel></op>
  <Var>customer</Var>
</Atom>

<Atom>
  <op><Rel>luxury</Rel></op>
  <Var>product</Var>
</Atom>

<Atom>
  <op><Rel>discount</Rel></op>
  <Var>customer</Var>
  <Var>product</Var>
  <Ind>7.5 percent</Ind>
</Atom>

<Atom>
  <op><Rel>spending</Rel></op>
  <Var>customer</Var>
  <Ind>min 5000 euro</Ind>
  <Ind>previous year</Ind>
</Atom>

<Atom>
  <op><Rel>premium</Rel></op>
</Atom>
<Var>customer</Var>

</Atom>

</Implies>

<Atom>

<op><Rel>luxury</Rel></op>

<Ind>Porsche</Ind>

</Atom>

<Atom>

<op><Rel>regular</Rel></op>

<Ind>Honda</Ind>

</Atom>

<Atom>

<op><Rel>spending</Rel></op>

<Ind>Peter Miller</Ind>

<Ind>min 5000 euro</Ind>

<Ind>previous year</Ind>

</Atom>

</Rulebase>

</Assert>