A Prototype Query-Answering Engine Using Semantic Reasoning

by

Kapil Dukle

Bachelor of Engineering

University of Mumbai, 1999

Submitted in Partial Fulfillment of the

Requirements for the Degree of Master of Science in the

Department of Computer Science and Engineering

College of Engineering and Information Technology

University of South Carolina

2003

Department of Computer Science
and Engineering
Director of Thesis

Department of Computer Science
and Engineering
2nd Reader

Department of Computer Science
and Engineering
3rd Reader

Dean of the Graduate School

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr Jose M. Vidal, for his constant help, patience and support throughout this thesis. I would also like to thank Dr Manton M. Matthews and Dr. Michael N. Huhns for being part of my thesis committee and providing guidance and suggestions during this project.

My appreciation also goes to staff of our department Jewel Rogers, Julie Neal, and Beverly Bradley for their help with administrative matters during my thesis work. I would also like to thank the members of the Jess mailing list, especially Mr. Ernest Friedman-Hill, for providing prompt and valuable suggestions during the project implementation phase.

I cannot end without thanking my family, on whose constant encouragement and love I have relied throughout my time at the university. Without their support and motivation, this work and my life would not be as they are today.

ABSTRACT

The Web represents an expansive medium for communications and information sharing. However, to date, the web has evolved as a medium composed of documents for human interpretation and consumption, rather than for software programs that can search and process the data automatically. Current web search engines use keyword-matching techniques to find key terms, and store them in the indexing database. They are however limited in their ability to retrieve relevant information due to the lack of semantics in interpreting these terms. The Semantic Web aims to change this. The vision is to markup pages with semantic languages such as the DARPA Agent Markup Language (DAML), and to link them in a way that can be automatically processed by software agents. Agents will be able to 'understand' the information and make meaningful inferences based on the meaning specified by the markup. This thesis describes a domain-independent Semantic Web information retrieval system that supports basic reasoning using DAML semantics and domain-specific rules. The system is built using the DAMLJessKB reasoning tool that uses the Java Expert System Shell (Jess) as the inference engine. We describe how these existing tools could be used to provide smarter, and more precise searching capabilities, and thus, move a step closer to fulfilling the Semantic Web vision.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Overview

The creation of the World Wide Web was a major step towards information sharing. It resulted in the development of a standard for information presentation, sharing, and retrieval, mainly for human interpretation and consumption. The current web is based on the use of the Hyper Text Markup Language (HTML) to display content. HTML provides programs, mainly Web browsers, with a set of instructions to present the information.

To make the web more useful, agents must be able to process the contents meaningfully. At present, finding information on web pages is based on syntax-oriented search – finding headers, keywords, or specific tags within a document to retrieve information. Users need to manually find the relevant information in the search results. The information (on the current Web) is not precise enough for computers to 'understand' and consume. In addition, extracting data from HTML documents is difficult, fragile and inefficient as there is no mechanic way to identify a useful pattern for extraction [32]. These extraction methods rely on the web page format to find information. If the source website decides to alter the web page structure, these techniques are rendered ineffective.

The Semantic Web approach is to add structure and semantics to the content of web pages. The idea is to weave a web that not only links documents to each other but also recognizes the meaning of the information in those documents – a task that people can ordinarily do well but is a tall order for computers [25]. The main goal is to facilitate information processing based on semantic content, enabling autonomous agent software to use this information to organize and filter data to meet the user's needs. Furthermore agents will be able to reason and make new inferences based on the semantics specified by the markup. The ability to deduce new knowledge from already specified knowledge would make them smarter and enhance their decision-making abilities.

The Semantic Web extends the current web by giving information a well-defined meaning, which enables computers and people to work in co-operation [1]. Software agents must have access to a set of inference rules and logic statements in order to be able to conduct information discovery, processing and reasoning.

However, it is not clear if the Semantic Web vision can be achieved with the current technologies or how hard it will be to develop the agents needed to support this vision. In this thesis we set out to build a prototype system which forms a microcosm of the Semantic Web goal. By building this system we hope to answer some fundamental questions about the Semantic Web and associated technologies:

1. Is the information currently available on the web suitable (after transformation) for use with the proposed ontology languages?

2. How hard is it to develop one of the required ontologies? What kind of information does the ontology need to have?

3. What other domain knowledge will our inference engine need?

4. What are the benefits/drawbacks of our final system over current keyword-based search engines?

The last question is the most important. Namely, we seek to ascertain the costs and benefits of migrating a sample website from the human-readable web to the Semantic Web. Is it really worth it?

We have developed a Semantic Web Query-Answering System to facilitate information retrieval within our domain using basic reasoning capabilities. DAML+OIL is used to generate ontologies for describing information because it is a formal knowledge representation language that enables automated inference. The formal semantics for DAML+OIL are fed in the form of rules to the DAMLJessKB reasoning tool. It utilizes Jess to carry out its reasoning. DAMLJessKB asserts the DAML+OIL semantics in the form of rules into the Jess knowledge base. In addition to the DAML+OIL rules, domain-specific rules are also supplied to the reasoning engine. This helps to make it more ‘aware’ of the class and property relationships, constraints, and other knowledge necessary to answer domain-specific queries in a smarter, more intelligent way. Jess uses these rules to carry out its reasoning, and answer queries that require knowledge beyond that present in the knowledge base.

1.2 Roadmap

The details of design, implementation, and testing our system will be presented in the following chapters. Chapter 2 gives a brief overview of the background and related technologies. The details of system design are covered in Chapter 3. Chapter 4 and Chapter 5 include the implementation and testing aspects of the system respectively. Chapter 6 describes the analysis and directions for future development.

CHAPTER 2

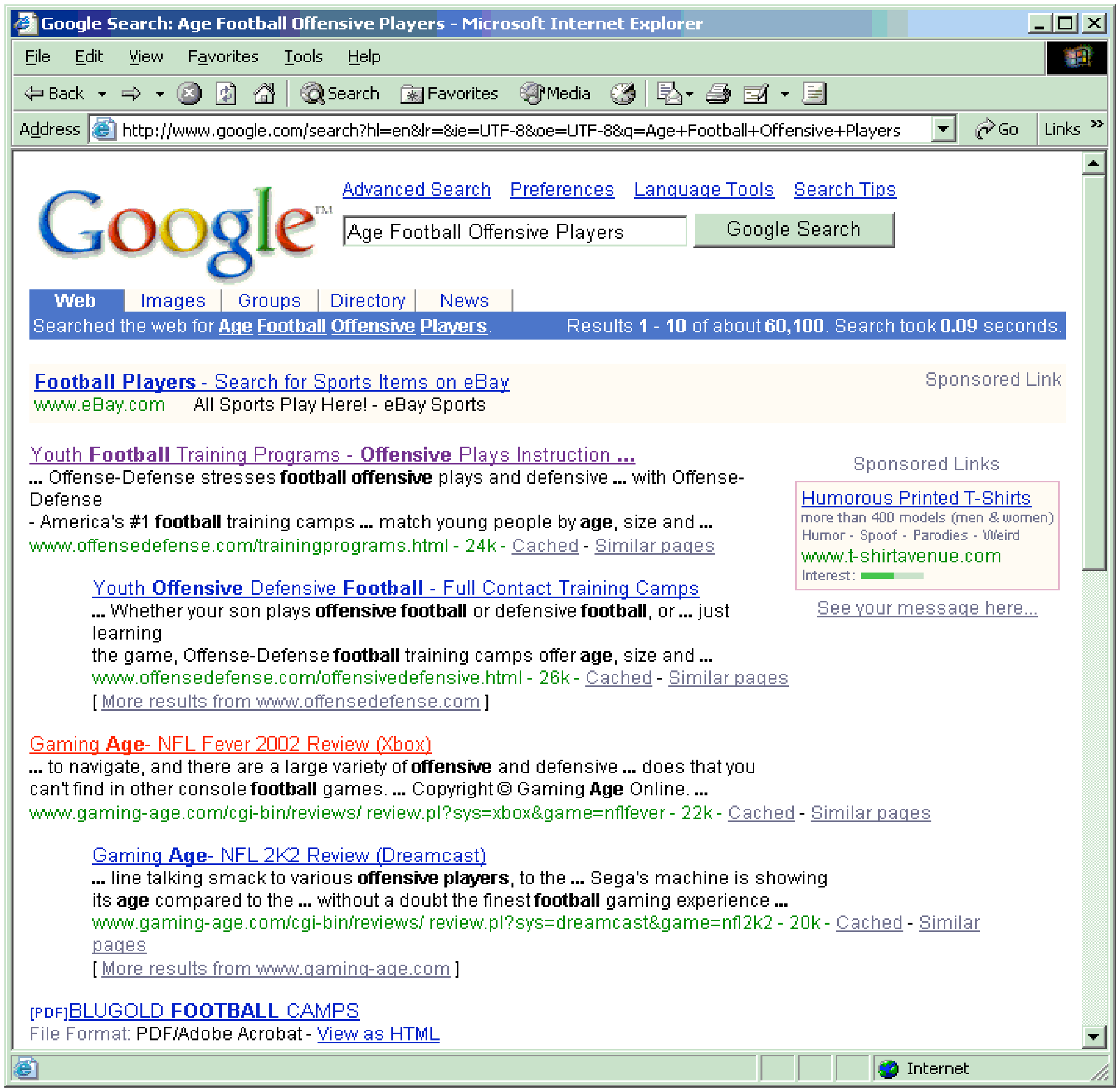
Background

2.1 Search Engines

Search engines are universally employed to find information on the Internet. Typically a search engine works by utilizing keyword matching to find key terms on web sites, and uses them to build the indexing database. The indexing database stores positional information (title, subheading, body of text) about the term, and assigns a weight to it, depending on its frequency on the web page. Each search engine has a different way of compiling the indexing database. Consequently, a search for the same word on different search engines may produce different lists. In response to a user query, the search engine matches the input with the index, and returns the results.

Consider an example to find the age of all Football Offensive Players. We used Google [31], a widely used search engine, for this purpose. The results are shown in Figure 2.1 We note that the results returned by the search contain too many irrelevant matches. The problem arises because the same word or phrase often refers to two or more unrelated contexts, which are all included in the results. In addition, search engines also tend to ignore relationships between keywords. This is evident from the results returned by the search engine.

The first result gives information about ‘Youth Football Training Programs’ simply because it finds our search keywords on that page. The page has no information about the age of football Offensive players. The second result is a review of a football video game. The search engine is thus unaware of the exact semantics relationship between our keywords, and returns a set of inaccurate results. More precise searches would have been possible if the search engine could understand the actual meaning of the keywords. The Semantic Web aims to change this.



Google Search: Age Football Offensive Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go Links
Advanced Search Preferences Language Tools Search Tips
Google Search
Web Images Groups Directory News
Internet

Address: http://www.google.com/search?hl=en&lr=&ie=UTF-8&oe=UTF-8&q=Age+Football+Offensive+Players

Age Football Offensive Players

Searched the web for **Age Football Offensive Players**. Results **1** - **10** of about **60,100**. Search took **0.09** seconds.

Football Players - Search for Sports Items on eBay
www.eBay.com All Sports Play Here! - eBay Sports
Sponsored Link

Youth **Football** Training Programs - **Offensive** Plays Instruction **...**
... Offense-Defense stresses **football offensive** plays and defensive **...** with Offense-Defense
- America's #1 **football** training camps **...** match young people by **age**, size and **...**
www.offensedefense.com/trainingprograms.html - 24k - Cached - Similar pages

Youth **Offensive** Defensive **Football** - Full Contact Training Camps
... Whether your son plays **offensive football** or defensive **football**, or **...** just learning
the game, Offense-Defense **football** training camps offer **age**, size and **...**
www.offensedefense.com/offensivedefensive.html - 26k - Cached - Similar pages
[More results from www.offensedefense.com]

Gaming **Age**- NFL Fever 2002 Review (Xbox)
... to navigate, and there are a large variety of **offensive** and defensive **...** does that you can't find in other console **football** games. **...** Copyright © Gaming **Age** Online. **...**
www.gaming-age.com/cgi-bin/reviews/ review.pl?sys=xbox&game=nflfever - 22k - Cached - Similar pages

Gaming **Age**- NFL 2K2 Review (Dreamcast)
... line talking smack to various **offensive players**, to the **...** Sega's machine is showing its **age** compared to the **...** without a doubt the finest **football** gaming experience **...**
www.gaming-age.com/cgi-bin/reviews/ review.pl?sys=dreamcast&game=nfl2k2 - 20k - Cached - Similar pages
[More results from www.gaming-age.com]

[PDF]BLUGOLD **FOOTBALL** CAMPS
File Format: PDF/Adobe Acrobat - View as HTML

Sponsored Links

Humorous Printed T-Shirts
more than 400 models (men & women)
Humor - Spoof - Parodies - Weird
www.t-shirtavenue.com
Interest:

See your message here...

Figure 2.1: Find the age of all Football Offensive Players

2.2 Ontologies

The Semantic Web relies heavily on formal ontologies to structure data for comprehensive and transportable machine understanding [24]. In AI, we attribute the notion of ontology to the specification of a conceptualization [2]. Ontologies for a specific domain are defined as the set of terms, inference rules, and relation between terms for that domain. In order to facilitate knowledge sharing and reuse between applications, ontologies must be defined in a machine-understandable format. An ontology typically contains a hierarchy of concepts within a domain and describes each concept's crucial properties through an attribute-value mechanism [3]. For example, the ontology of a university includes professors, students, courses offered by professors, major programs in the university, rules such as prerequisites for a particular course, number of courses required for graduation, and so forth.

An important step in realizing the Semantic Web is to link websites to ontologies. Agents will use rules and concepts described by the ontologies to gather, process and exchange information from web pages. Creating links between ontologies enables knowledge reuse in the sense that an ontology need not redefine a concept found in another. Linking ontologies represents a 'distributed knowledge base' setting, and using them in this manner, promotes common understanding of concepts, and resolves ambiguity resulting from similar terminology. They can be used in a simple fashion to improve the accuracy of Web searches—the search program can look for only those pages that refer to a precise concept instead of all the ones using ambiguous keywords [1]. Ontologies, thus, form an important part of the Semantic Web search.

2.3 Semantic Web Languages

2.3.1 XML

The goal of the Semantic Web is to add structure and semantics to web pages in a way that promotes its understanding by machines. To date, information on the web is encoded using HTML, which focuses on how data is displayed for human interpretation. The past few years have witnessed XML (eXtensible Markup Language) as a popular language for storing and exchanging data between machines. The word 'Extensible' suggests that XML is actually a data format that allows users to specify their own tags. XML enforces a strict syntax for documents, but does not imply a specific interpretation of the data [4]. For example, the tag 'chair' could either refer to the inanimate object - chair, or to the head of a company. In such cases, namespaces are used, simply to distinguish similar names used in XML documents. XML, however, cannot be used to represent complex knowledge due to its inability to specify semantics. Specifications like DTD (Document Type Definition) and XML Schema have been employed to describe XML data structures – the names of elements and attributes, and their use in documents [4]. DTD and XML Schema however do not specify data meaning, and the use of XML as a semantic language leaves much to be desired.

Namespaces are also employed in the XML Schema specification. An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names [5]. The names defined in an XML schema belong to a target namespace. Declarations and terms in an XML schema can refer to names belonging to other namespaces. An XML Schema fragment with the namespace declaration is shown in Figure 2.1.

```
<xsd:schema
        targetNamespace = "http://www.thisexample.com/ex1"
         xmlns:xsd = "http://www.w3.org/1999/XMLSchema">

   <xsd:element  name = "SSN"  type = "xsd:positive-integer" />
```

Figure 2.2: XML Schema Namespace Declaration

All elements prefixed with the `xsd:` should be understood as referring to terms from the `http://www.w3.org/1999/XMLSchema` namespace. In the fragment, the elements 'schema', 'element', and 'positive-integer' belong to the namespace `http://www.w3.org/1999/XMLSchema`

2.3.2 RDF

RDF (Resource Description Framework) [26] is a model for representing data about resources on the Web. It is built on top of XML, and represents data in the form of triples – resource, property and statement. For example, the statement 'The author of this paper is Kapil Dukle' has the following triples: 'Kapil Dukle' – statement, 'author' – property, and 'paper' – resource. The RDF model does not sufficiently define the semantics of the application domain. It just provides a domain-neutral mechanism to describe metadata [4]. RDFS (RDF Schema) is a type system for RDF. It allows you to define class hierarchies, specify properties, and enforce domain and range specifications for these properties. However, RDF and RDFS are still limited as a knowledge representation language due to their lack of support for advanced features like defining properties of properties (unique, transitive, inverse), disjoint classes, and so forth.

2.3.3 DAML

DAML (DARPA Agent Markup Language) [12] is a semantic markup language aimed at providing machine-interpretable information on the Internet. The DAML program was founded by a group of researchers in August 2000 in Boston. DAML is an attempt to address shortcomings of the RDF and RDFS specification by incorporating additional semantic features. DAML along with OIL (Ontology Inference Layer) can be used to describe ontologies. DAML+OIL includes support for classifications, property restrictions, and facilities for data typing based on type definitions provided in the W3C XML Schema Definition Language. Because of these added semantics, if you tell a computer something in DAML, it can give you new information, based entirely on the DAML standard itself. DAML+OIL gives computers one extra small degree of autonomy that can help them do more useful work for people [6].

Consider the following example illustrated in [6]:

Given the following DAML statements:

```
(motherOf subProperty parentOf)
(Mary motherOf Bill)
```

when stated in DAML, allows you to conclude:

```
(Mary parentOf Bill)
```

based on the logical definition of `subProperty` as given in the DAML specification.

DAML+OIL provides a basic foundation for computers to make the same sort of inferences that humans do. It is thus a semantically rich language, well suited for the fulfilling the Semantic Web goal. DAML can dramatically improve traditional ad hoc information retrieval, as its semantics will improve the quality of retrieval results [7].

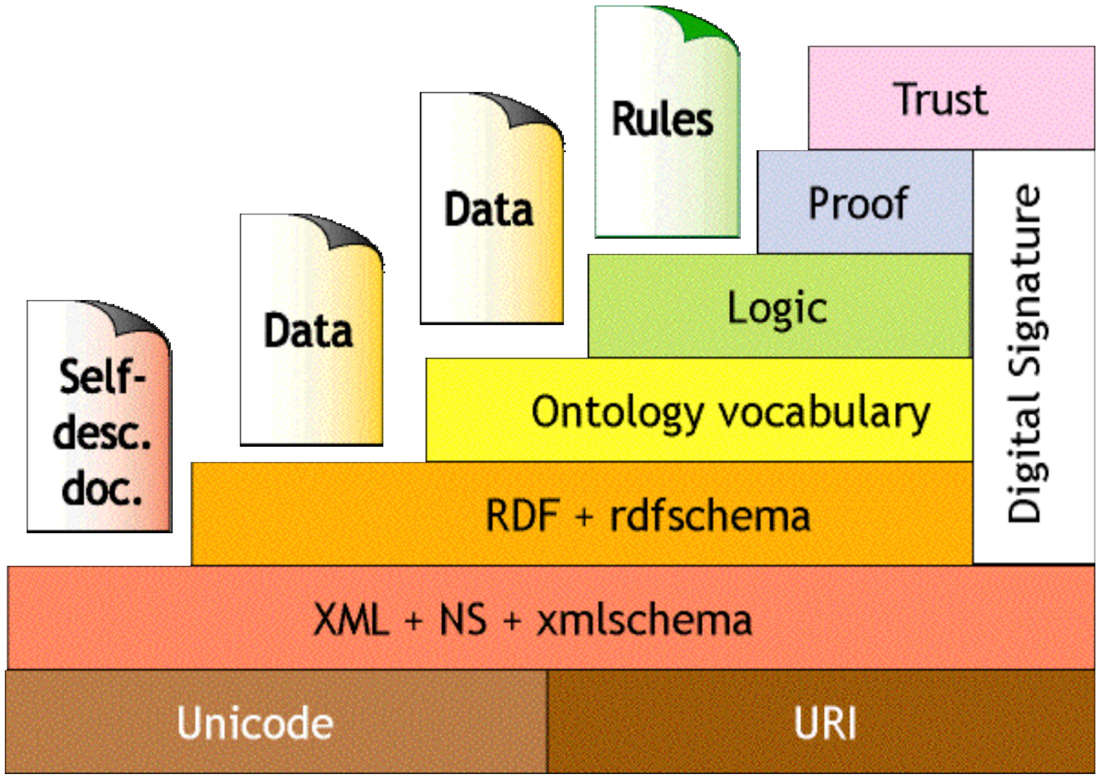


Figure 2.3: Layers of the Semantic Web ([27])

According to Tim Berners-Lee, the lower layers of the Semantic Web, like XML, are already pretty much standardized. The middle layers, like RDF and Web ontology, are in the process of being standardized. And the upper layers begin things like ontology vocabulary and logic are still in the research stage. DAML+OIL, built on top of RDF, is among the upper layers that are part of active research.

This paper describes the use of DAML+OIL to construct ontologies for a particular domain, and markup instance data. Our reasoning tool DAMLJessKB uses this DAML+OIL markup to assert semantics, and facts into the inference engine.

2.4 The Inference Engine: Jess

Jess (Java Expert System Shell) is a rule-based Expert System Shell written entirely in Sun Microsystems' Java language. It was developed by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. Jess was initially developed as a Java version of the CLIPS (C Language Integrated Production System) expert system shell, but it has now grown into a complete, distinct, dynamic environment of its own. It provides a convenient way to integrate complex reasoning capabilities into Java-based software. The Jess language includes many elements such as negation, and combinations of boolean conjunctions and disjunctions to specify complex rules, facts and queries. These features add to the reasoning capabilities of Jess and make it suitable for Semantic Web reasoning.

The purpose of Jess is to continuously apply a set of rules to a collection of facts stored in the knowledge base. Often the rules will represent the heuristic knowledge of a human expert in some domain, and the knowledge base will represent the state of an evolving situation (an interview, an emergency) [8]. Rules that apply are fired, or executed. Jess uses a very efficient method known as the Rete algorithm to match the rules to the facts. Rete makes Jess much faster than a simple set of cascading if.. then rules in a loop.

2.4.1 Rete Algorithm

Following is a brief description of Rete taken from [9]:

> "The typical expert system has a fixed set of rules while facts in the knowledge base changes continuously. However, it is an empirical fact that, in most expert systems, much of the knowledge base is also fairly fixed from one rule operation to the next. The obvious implementation would be to keep a list of the rules and continuously cycle through the list, checking each one's left-hand-side (LHS) against the knowledge base and executing the right-hand-side (RHS) of any rules that apply. This is inefficient because most of the tests made on each cycle will have the same results as on the previous iteration. Most of the tests will be repeated. The computational complexity is of the order of O(RF^P), where R is the number of rules, P is the average number of patterns per rule LHS, and F is the number of facts on the knowledge base. This increases dramatically as the number of patterns per rule increases.
>
> The Rete algorithm improves this by remembering past test results across iterations of the rule loop. Only new facts are tested against any rule LHSs. Additionally, new facts are tested against only the rule LHSs to which they are most likely to be relevant. As a result, the computational complexity per iteration drops to something more like O(RFP), or linear in the size of the fact base.

The Rete algorithm is implemented by building a network of nodes, each of which represents one or more tests found on a rule LHS. This network of nodes processes facts that are being added to or removed from the knowledge base. At the bottom of the network are nodes representing individual rules. When a set of facts filters all the way down to the bottom of the network, it has passed all the tests on the LHS of a particular rule and this set becomes an *activation*. The associated rule may have its RHS executed (*fired*) if the activation is not invalidated first by the removal of one or more facts from its activation set ."

2.4.2 Jess Constructs

A few JESS constructs used by our system are described below:

Facts

A rule-based system such as Jess maintains a collection of facts known as the knowledge base. It is similar to a relational database, especially in that the facts must have a specific structure. In Jess, there are three kinds of facts: ordered facts, unordered facts, and definstance facts.

Ordered facts are simply lists, where the first field (the head of the list) acts as a sort of category for the fact.

This application asserts ordered facts (using DAMLJessKB) into the knowledge base. An ordered fact is shown.

```
(PropertyValue  rdf:type  daml:Thing  daml:Class)
```

Defrules

The knowledge base contains a collection of facts. The `defrule` construct enables Jess to carry out actions based on one or more facts. In addition, the knowledge base can also be queried to find relationships between facts.

A Jess rule is similar to an if...else statement in a procedural language. The difference lies in the way these statements are executed. While if... then statements are executed at a specific time and in a specific order, according to how the programmer writes them, Jess rules are executed whenever their if parts (their left-hand-sides) are satisfied, given only that the rule engine is running [8].

A `defrule` construct in Jess is demonstrated by the following example. The rule states that instances of a subclass are instances of a class. It is a snippet from the DAML+OIL rules file supplied with DAMLJessKB [10].

```
(defrule subclassInstances
      "An instance of a subclass is an instance of the parent
      class. This enforces and makes meaningful the
      daml:subClassOf  relationship"

      (PropertyValue daml:subClassOf   ?child     ?parent)
      (PropertyValue rdf:type          ?instance  ?child)
      =>
      (assert
       (PropertyValue rdf:type        ?instance   ?parent)
      )
)
```

The `subclassInstances` rule will be activated when the two facts appear in the knowledge base, i.e. when a child that is a subclass of the parent, and an instance that is a subclass of child is found. The result of the rule activation is that a fact stating the resulting instance is a type of the parent class is asserted. Ordered facts can be added to the knowledge base using the `assert` function.

Defqueries

The `defquery` construct lets you create a special kind of rule with no right-hand-side. While rules act spontaneously, queries are used to search the knowledge base under direct program control [8]. A rule is activated once for each matching set of facts, while a query gives you a `java.util.Iterator` of all the matches.

```
(defquery defensivePlayers
      "Find all values having type football:DefensivePlayer"
      (PropertyValue rdf:type ?n football:DefensivePlayer)
)
```

The query has an internal variable `?n`. It can be run using the `run-query` command. A list of all facts in the knowledge base that contain objects with a `rdf:type` having value `football:DefensivePlayer` is returned.

Run-query command

The `run-query` command lets you supply values for the external variables (if any) of a query and obtain a list of matches. It returns a `java.util.Iterator` of `jess.Token` objects for each matching combination of facts.

A detailed explanation of all Jess terms and constructs can be found in the Jess Language section at [8].

2.5 The Reasoning Tool: DAMLJessKB

DAMLJessKB is a tool for reasoning with the Semantic Web. Joe Kopena developed DAMLJessKB at the Intelligent Time-Critical Systems Laboratory at Drexel University. It is a description logic reasoner for performing inference with the DARPA Agent Markup Language (DAML). It is currently used in a various research projects like: Penn State University's Industrial Engineering Department, UMBC Agents group, Coca Cola (enterprise integration), SRI, and Lockheed Martin's Advanced Technology Laboratories [11].

Creating Semantic Web applications requires applying the semantics of these languages to make decisions based on the inferences entailed by the formal definitions of these languages [11]. DAMLJessKB utilizes JESS to carry out its reasoning.

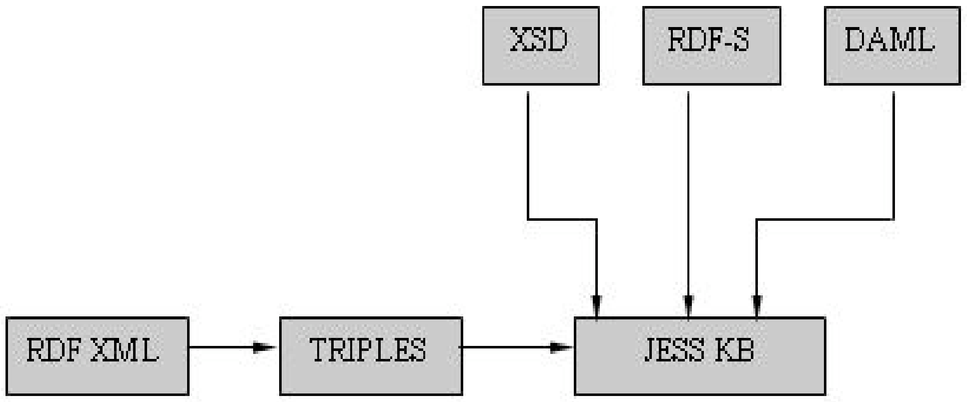


Figure 2.4: Overview of the DAMLJessKB Process [11]

Figure 2.3 shows the overview of the DAMLJessKB process. DAMLJessKB uses ARP (Another RDF Parser) to load and parse RDF documents. ARP is a part of the Jena Semantic Web toolkit developed at Hewlett Packard labs [13]. ARP produces a stream of RDF triples to be asserted into Jess, the inference engine. DAMLJessKB assumes a closed world system (i.e., if a fact does not exist it is assumed to be false) [11]. The main purpose of DAMLJessKB is to provide an interface for asserting facts from RDF documents into Jess and applying DAML+OIL semantics.

Triples generated by ARP are collected and simple transformations are applied. The most basic of these transformations is to translate Uniform Resource Identifiers (URIs) into valid Jess symbols by removing invalid characters. For example, tildes are replaced by a predefined escape code. DAMLJessKB also inserts the dummy predicate `PropertyValue` in front of each triple before asserting into Jess. For anonymous objects, a unique identifier is generated by ARP and the unary predicate 'anonymous' is asserted.

```
(PropertyValue
     http://www.w3.org/1999/02/22-rdf-syntax-ns#type
    http://www.daml.org/2001/03/daml+oil#Thing
    http://www.daml.org/2001/03/daml+oil#Class)
```

The set of rules implementing DAMLJessKB's reasoning can be divided into two categories [11]. One concerns reasoning on instances of classes. The second concerns terminological reasoning, determining relationships between the classes themselves. An example of a basic inference rule for instances is "Any instance of a subclass is an instance of the parent class". The second example involves terminological reasoning. For a class is defined as the intersection of a set of classes (conjunction), a common terminological inference is subsumption between such intersections. The rule states "A class composed of the intersection of a set of classes is a subclass of a class composed of the intersection of a subset of those classes or subclasses of those classes". In the case where the two intersections are equivalent, each class will be asserted as a subclass of the other [11].

Our system uses DAMLJessKB to assert DAML+OIL semantics and other domain-related information in the form of facts and rules into Jess.

2.6 Jena Semantic Web Toolkit

Jena toolkit [13] was developed at Hewlett Packard labs as part of their Semantic Web research program. It is Java API for manipulating RDF documents.

Features of the toolkit include the following [13]:

- statement centric methods for manipulating an RDF model as a set of RDF triples
- resource centric methods for manipulating an RDF model as a set of resources with properties
- cascading method calls for more convenient programming
- built in support for RDF containers - bag, alt and seq
- enhanced resources - the application can extend the behaviour of resources
- integrated parsers (ARP and David Megginson's RDFFilter)

The toolkit supports parsing of DAML ontologies. This application mainly uses the Jena toolkit to load and parse a DAML ontology given a URI. In addition, it uses Jena to traverse class and property hierarchies within the ontology, and determine property types (whether `daml:ObjectProperty` or `daml:DatatypeProperty`). This is useful for loading ontologies dynamically, and making appropriate selections (for the semantic search) using the application's graphical user interface.

In general, the Jena toolkit includes the following DAML functionality [13]:

- load a DAML ontology document given a URL
- detect embedded `daml:Ontology` elements that import other ontologies, and load those too (can be selectively or completely disabled)
- all information is actually stored in a Jena model, so full access is always available to the RDF form of the ontology
- Java classes that shadow the components of the DAML vocabulary (class, property, list, etc) and allow direct programmatic access to the various properties that they support (e.g. `disjointWith` on a class)
- traversal of sub-class and sub-property hierarchy
- programmatically create or update any DAML value
- write ontology out to a file or stream
- iterators for classes, instances and properties that understand the class/property hierarchies

2.7 Java Swing HTML parser

This application utilizes the Swing HTML parser to create the Scraper program in order to obtain real-time instance information for our particular domain - football. Specifically it reads and parses information about football players, football games, and teams from the NFL website [14] and annotates it using DAML. We created an API to generate DAML instance markup for each class in the football ontology. This information will be required to answer queries related to our domain.

Writing a parser for HTML is a very difficult task. The problem with HTML is that although there is an HTML specification, in practice it is rarely followed by web designers or browser vendors [15]. The specification itself is very flexible and does not enforce strict syntax for HTML documents.

For example, element tags in HTML may be uppercase, lowercase or mixed case.
`<html>, <HTML>, and <HtmL>` are all valid HTML tags.
Attribute values may or may not be quoted.
`<font face="Arial">` and `<font face=Arial>` are both correct.
Usually tags in HTML come in pairs. Every start tag has a corresponding end tag.
`<HTML>` (Start tag) and `</HTML>` (End tag).
However, some tags like `<P>` (Paragraph tag) have an optional end tag. It is not an error to omit the `</P>` end tag in HTML code.

Such irregularities make it difficult to write an HTML parser that covers all such possibilities. Fortunately starting JFC 1.1.1, Sun Microsystems provides classes for basic HTML parsing in the `javax.swing.text.html` and `javax.swing.text.html.parser` packages [15].

Essential Classes:

`javax.swing.text.html.HTMLEditorKit.ParserCallback`

The `ParserCallback` class is a public inner class inside

`javax.swing.text.html.HTMLEditorKit`.

The standard implementation of this class contains six callback methods that do nothing.

```
public void handleText(Char[] text, int position)

public void handleComment(Char[] text, int position)


public void handleStartTag(HTML.Tag tag, MutableAttributeSet
                                      attributes, int position)

public void handleEndTag(HTML.Tag tag, int position)

public void handleSimpleTag(HTML.Tag tag, MutableAttributeSet
                                      attributes, int position)

public void handleError(String errorMessage, int position)
```

This class has to be subclassed in order to provide implementation. To parse an HTML file, you write a subclass of `HTMLEditorKit.ParserCallback` that responds to text and tags as required.

There's also a `flush()` method to perform final cleanup. This method is invoked after the parser has finished parsing the document.

```
public void flush() throws BadLocationException
```

`javax.swing.text.html.HTMLEditorKit.Parser`

This is the main parsing class. Since this is an abstract class, the actual parsing is done by an instance of the concrete subclass

`javax.swing.text.html.parser.ParserDelegator`.

```
public class ParserDelegator extends HTMLEditorKit.Parser
                                  implements Serializable
```

The `ParserDelegator` class must be configured with a DTD using the protected, static methods:

```
protected static void setDefaultDTD()

protected static DTD createDTD(DTD dtd, String name)
```

The parse method in the `ParserDelegator` class parses the HTML document.

```
public void parse(Reader r, HTMLEditorKit.ParserCallback cb,
                       boolean ignoreCharSet) throws IOException
```

The basic parts of an HTML document are start tags, end tags, empty tags, text, and comments. The parse method reads an HTML document from a `Reader` object, and looks for these parts in the document. Each time the parser encounters one of these items, it invokes the corresponding callback method from the `HTMLEditorKit.ParserCallback` instance.

An instance of the `HTMLEditorKit.ParserCallback` subclass is passed to the `HTMLEditorKit.Parser's parse()` method, along with the `Reader` object from which HTML will be read. The actual parsing is handled inside the callback methods in the `HTMLEditorKit.ParserCallback` subclass. The `parse()` method simply reads the entire HTML document using the `Reader` object, and each time it sees a tag, comment, or text block it invokes the appropriate callback method in the `HTMLEditorKit.ParserCallback` instance. Parsing takes place in a separate thread, so the `parse()` method usually returns before parsing is completed.

2.8 Related Projects

2.8.1 ITTalks Project at UMBC

The ITTalks website [16] conveys information about various IT events and seminars, including topic, speaker and location. It was developed at the University of Maryland, Baltimore County, as part of their DAML project. The project facilitates user and agent interaction for locating talks on information technology [16].

ITTalks uses DAML+OIL for knowledge base representation, reasoning and agent communication [16]. A set of ontologies describes events like talks, and related information like places, speakers, topics and schedules. DAML+OIL is used to mark up this information. Additionally, ITTalks uses DAML+OIL (for queries and notifications) as content in the Agent Communication Language (ACL). It identifies an agent that provides Semantic Web services. The agent is a front end for the system.

In order to extend its capabilities, the research group at UMBC has been working on DAML-based reasoning to carry out tasks like interest-based talk matching and agent Communication language manipulation/understanding [19]. Two DAML-based inference engines have been developed.

The first Inference Engine (IE) uses SWI-Prolog [17], with the SWI-Prolog SGML parser as a DAML parser. The second one uses XSB [18] as an Inference Engine, Yajxb as the XSB Bridge to the Java language, and the RDF API as a DAML parser.

Interest-based matching works as follows [19]:

- The user first submits a query at the IITalks website
- The query includes:
    - The URI of the user's profile (a DAML file)
    - The number of desired results
- A search description (a DAML file) is generated, based on `Search.daml`, which defines the search ontology

- The search description is sent to the IE, which parses the DAML file, reads the user profile (through the HTTP) interface, reads the profile ontology DAML file, and parses them into triples
- The IE reads the Topic ontology DAML file, gets all the talks from `www.ittalks.org` (through the HTTP) interface, and parses them into triples
- The IE does an interest match, computing the distance between user interests and talk topics, and generating an interest level for each
- Interest levels are ranked, revealing the talks of greatest interest to the user (the result number is specified in the query DAML file) , generate result DAML file which follow result ontology defined by `result.daml`
- Results are returned to the user

2.8.2 Stanford Knowledge Systems Laboratory (KSL)

KSL is developing technology for reasoning with knowledge expressed in DAML on distributed Web sites. The research focus at KSL is in the following areas: DAML Language Research, DAML-Enabled Web Services, DAML Tools Research and DAML-Based Query Answering [20].

The team at Stanford university are addressing both the standard issues about how to reason effectively with knowledge expressed in an object-oriented language augmented with rules and the issues raised by the knowledge using ontologies resident on (perhaps multiple) other Web sites [20]. The technology includes a DAML reasoner called JTP (Java Theorem Prover) implemented in Java that contains a general-purpose theorem prover integrated with a collection of special-purpose reasoners designed specifically for DAML+OIL and specific task domains.

2.8.3 OWLIR Information Retrieval System at UMBC

OWLIR [28] is a system to retrieve documents that contain both free text and semantically enriched markup documents. It is intended to provide a framework, which is able to extract and exploit the semantic information from these documents, perform sophisticated reasoning and filter results for better precision [28].

OWLIR consists of two main components: a set of ontologies describing their domain of interest and a hybrid information retrieval mechanism. The goal of their ontology development is to develop an ontology which will help users interested in different events in the university, retrieve relevant information [28]. Ontologies are encoded using DAML+OIL markup.

The event announcements are described in free text. The AeroText [29] system was utilized for text extraction of key phrases and elements from free text. DAML generation components developed at UMBC translate this extraction results into a RDF triple model that utilizes the DAML+OIL syntax. OWLIR uses DAMLJessKB to carry out its reasoning. DAMLJessKB provides basic facts and rules that facilitate drawing inferences on relationships such as Subclasses and Subproperties [28]. The inferential capabilities of DAMLJessKB are enhanced by augmenting existing DAML semantics with domain specific rules. Triples generated by DAMLJessKB form the knowledge base of

HAIRCUT [30] . HAIRCUT (Hopkins Automated Information Retriever for Combing Unstructured Text) is an information retrieval system developed at John Hopkins University, Applied Physics Lab. HAIRCUT retrieves information in response to queries specified using DQL (DAML+OIL Query Language). This framework advocates the interdependency of search and inference for precise retrieval over semantic content.

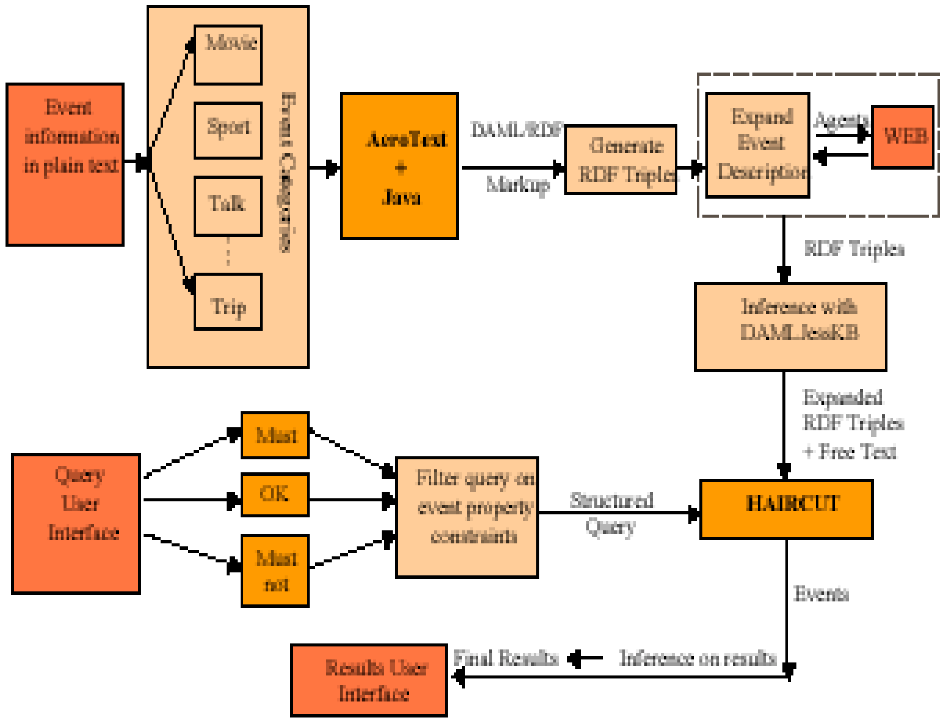


Figure 2.5: OWLIR Process Flow [28]

CHAPTER 3

Architecture

3.1 Overview

The Semantic Web aims to add semantics and better structure to the information available on the web. Underlying this is the goal of making the web more effective for humans. The basic idea is to create an environment for intelligent programs to carry out tasks independently on behalf of the user. In addition to the automation resulting from this approach, we also need to be able to obtain smart, relevant answers to our searches on the web.

Thus, the goal of the Semantic Web is to provide a wide range of functionality on the Web, using varied tools such as scraper programs for information retrieval, and reasoning engines to provide 'smart' query-answering services.

A wide range of Semantic Web languages such as RDF, DAML and OIL have been proposed. In addition, a number of independent tools and APIs - RDF parsers, DAML markup tools, and Ontology editors that implement these technologies have been developed. Yet, the Semantic Web has been slow to materialize.

This application demonstrates one approach to building the Semantic Web. It integrates existing technologies to provide a search service for our chosen domain. Currently though search engines do a reasonable job of retrieving information on the web, a number of those results may not be pertinent to the user's needs at all. The user needs to manually look for the relevant information in the search results. The information on the web pages is not precise enough for computers to 'understand' and consume.

The emergence of DAML as a markup language helps alleviate some of these problems. For the Semantic Web to be successful, web pages need to contain information marked up using a semantic language such as DAML, in addition to HTML (Hyper Text Markup Language). Information in these documents is organized by describing individual objects in the domain, their relationships and properties.

This application focuses on the use of DAML language, and DAML-enabled technologies to provide smarter search results based on the semantics defined by the language, and the reasoning capabilities provided by Jess and DAMLJessKB.

3.2 System Components

We have build a Semantic Web Query-Answering (QA) system that supports advanced querying, and information retrieval with a certain amount of reasoning capability within our domain. The application answers queries pertaining to the football domain – our domain of choice for this system. Although we have chosen a particular domain for our system, the system is domain-neutral - it is not tied to a single ontology or domain. It can be used as a query answering system for any domain described by a valid DAML ontology and containing DAML instance values.

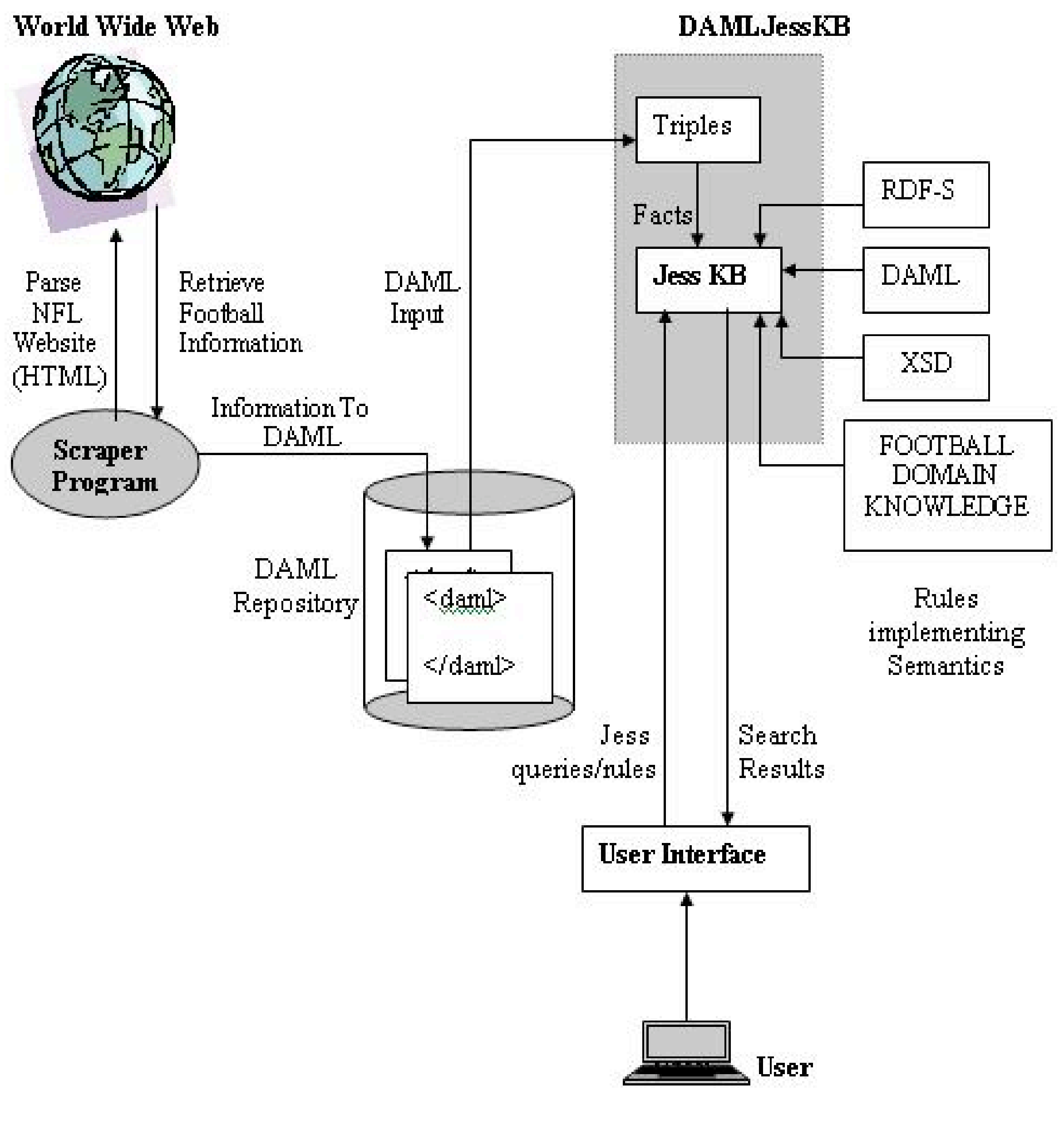


Figure 3.1: Architecture of the Semantic Web Query-Answering System

Figure 3.1 shows the architecture of our system. As noted previously, we have chosen football as the domain for this application. The basis of the system is the set of ontologies used to define entities in our domain, and DAMLJessKB, the reasoning tool. Ontologies are described using DAML. These ontologies not only provide class, property and relationship definitions but also provide a structure to mark up the instance data used to answer queries. They are also used to perform basic consistency checking [21] of the instance data – to check whether the DAML instance data conforms to the ontology specifications.

Our architecture identifies three main components:

1. Scraper program

The instance information for the football domain is available at the NFL website [14]. The information, however, is marked up using HTML. The primary aim of this scraper agent is to parse HTML content, and mark up the information into DAML pages. The Scraper program may be run periodically to ensure the DAML instance information required by the search system is kept current.

2. DAMLJessKB

DAMLJessKB is used to carry out automated reasoning specified by the DAML semantics. DAMLJessKB internally uses Jess as the inference engine. Given any RDF or DAML+OIL document, the information can simply be viewed as a set of triples. Each triple is basically a relationship between a Subject and Object through a Predicate. DAMLJessKB reads DAML documents and generates a stream of triples for assertion into the production system – Jess. Rules derived from the semantics of DAML+OIL are applied to populate the knowledge base with additional facts that can be entailed from the input [11]. One advantage of this approach is that the information extracted from the input documents, as well as the new inferred information is immediately available to Jess.

DAMLJessKB, thus, acts as an interface for the Jess object that contains the knowledge base. Its primary purpose is to supply an interface for loading RDF or DAML documents into Jess and automatically applying RDF/RDF-S/XSD/DAML+OIL semantics [11].

3. Jess Inference Engine

DAMLJessKB internally uses Jess as the inference engine. The Jess object containing the knowledge base performs the reasoning essential to this application. It accepts facts supplied in the form of triples by DAMLJessKB and loads them into its knowledge base. DAMLJessKB asserts rules specifying DAML+OIL semantics into Jess. In addition, rules pertinent to our domain (football) are also loaded. This helps it understand our domain better and to makes it better equipped to handle complex queries related to football.

3.3 Design Consideration

3.3.1 DAML

The goal of DAML program is to transform the current human-driven web into one that promotes human-machine interaction, thus enabling machines to make decisions on behalf of the user. This can only be achieved by using semantic tags to mark up information on the web. The emergence of DAML as a semantic language is an important step in this direction.

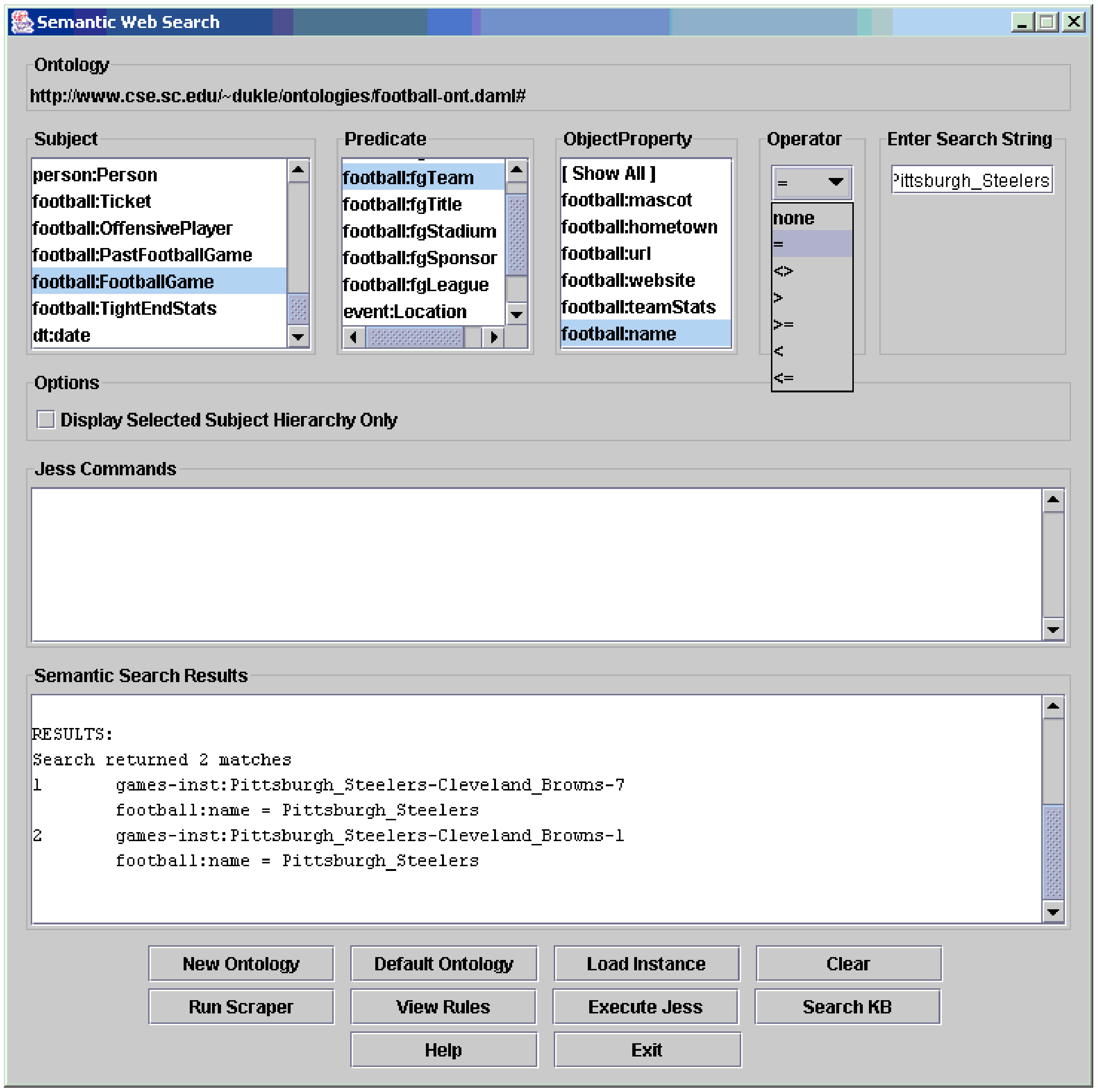
Two existing languages for the Semantic Web have already been developed – XML (eXtensible Markup Language) and RDF (Resource Description Framework). DAML, however, provides a number of advantages over these markup approaches. It enables semantic interoperability— instead of enabling only syntactic interoperability as is done in XML [22]. RDF and RDFS, built on top of XML, are frameworks for describing and interchanging metadata. They however lack advanced features like data types, expression for enumerations, properties of properties (unique property, inverse property), and disjoint classes. DAML provides these facilities (lacking in RDF) and additional semantics such as restrictions and cardinality constraints. Thus, DAML being a semantically rich language, is our choice as the markup language for this application.

3.3.2 DAMLJessKB

We have chosen DAMLJessKB to incorporate basic reasoning capabilities in our application because of its strong support for DAML semantics. DAMLJessKB supports a broad range of standards like the XSD, RDFS and DAML+OIL. It implements a large portion of the existing DAML+OIL semantics including XML Schema datatypes, and the `equivalentTo` and `TransitiveProperty` elements. In addition, DAMLJessKB is implemented using Java, and that makes it easier to integrate it with existing Java applications. DAMLJessKB differs from most Semantic Web reasoners, which are built on description logic and general theorem provers. It assumes a closed world assumption (i.e., if a fact does not exist it is assumed to be false), which is often a reasonable assumption in practice [11].

3.3.3 Jess

DAMLKessKB uses Jess as the inference engine. Jess is a very powerful scripting language with a great degree of expressiveness. The Jess rule language includes elements not present in many other production systems, such as negation and arbitrary combinations of boolean conjunctions and disjunctions [11]. It includes a wide range of constructs like functions, templates, facts, rules and queries, and powerful pattern matching techniques necessary for building intelligent systems. Our system will benefit immensely from the use of these advanced pattern-matching mechanisms. These techniques will be immensely useful for retrieving information in response to advanced and more complex search queries. Jess supports the development of rule-based expert systems that can be integrated with existing Java code.



http://www.cse.sc.edu/~dukle/ontologies/football-ont.daml#

Subject: person:Person, football:Ticket, football:OffensivePlayer, football:PastFootballGame, football:FootballGame, football:TightEndStats, dt:date

Predicate: football:fgTeam, football:fgTitle, football:fgStadium, football:fgSponsor, football:fgLeague, event:Location

ObjectProperty: [Show All], football:mascot, football:hometown, football:url, football:website, football:teamStats, football:name

Operator: = (none, =, <>, >, >=, <, <=)

Enter Search String: Pittsburgh_Steelers

```
RESULTS:
Search returned 2 matches
1       games-inst:Pittsburgh_Steelers-Cleveland_Browns-7
        football:name = Pittsburgh_Steelers
2       games-inst:Pittsburgh_Steelers-Cleveland_Browns-1
        football:name = Pittsburgh_Steelers
```

Figure 3.2: Semantic Web Query-Answering System

CHAPTER 4

Implementation

4.1 Ontology creation

Ontologies serve as metadata schemas, providing a controlled vocabulary of concepts, each with explicitly defined and machine-processable semantics [24]. DAML is used to create our ontologies that define entities, properties, and relationships in the football domain. In addition, they also provide a structure to marking up the instance data that will be used to answer queries.

We have developed an ontology that will help users retrieve relevant information about football. Entities in our domain include `FootballPlayer`, `FootballGame`, and `FootballTeam`. They are characterized by properties, for example, (`name, age, team`) for a `FootballPlayer`. They also have relationships with other entities; for example, a `FootballPlayer` belongs to a `FootballTeam`.

We have developed a general ontology that describes the football domain. A snippet of the DAML football ontology is shown in Figure 4.1. The Appendix section describes the complete ontology.

```
<daml:Class rdf:ID="FootballGame">
 <rdfs:label>Football Game</rdfs:label>
 <rdfs:subClassOf
   rdf:resource="http://daml.umbc.edu/ontologies/talk-ont#Event"/>
</daml:Class>

<daml:Class rdf:ID="FootballPlayer">
 <rdfs:label>FootballPlayer</rdfs:label>
 <rdfs:subClassOf
"http://www.cse.sc.edu/~dukle/ontologies/player-ont.daml#Player" />
</daml:Class>

<daml:Class rdf:ID="OffensivePlayer">
 <rdfs:label>OffensivePlayer</rdfs:label>
 <rdfs:subClassOf rdf:resource="#FootballPlayer" />
</daml:Class>

<daml:DatatypeProperty rdf:ID="fpTeam">
 <rdfs:comment> Player can belong to only one team. Hence, team is a
UniqueProperty. </rdfs:comment>
 <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProp
 />

<rdfs:domain rdf:resource="#FootballPlayer" />
<rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"
 />
</daml:DatatypeProperty>
<daml:DatatypeProperty rdf:ID="yearsPro">
 <rdfs:comment> yearsPro has at most one value</rdfs:comment>
 <rdfs:domain rdf:resource="#FootballPlayer" />
 <rdfs:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"
/>
</daml:DatatypeProperty>

<daml:Class rdf:about="#FootballGame">
 <rdfs:subClassOf>
  <daml:Restriction daml:cardinality="2">
   <daml:onProperty rdf:resource="#fgTeam" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>
```

Figure 4.1: DAML Football Ontology

The football ontology supports interoperability with other ontologies, enabling knowledge reuse. A `FootballGame` has properties begin time, end time, and place that are essentially properties of an `Event`.The `Event` class is defined by the UMBC Talk Ontology [23]. We define `FootballGame` to be a subclass of `Event`, and in a way, link to the talk ontology described by UMBC [23]. This promotes common understanding of concepts, and avoids redefinition of terms defined in other ontologies. Ontology sharing is an important aspect of the Semantic Web because of the heterogeneity among domains on the web. Some amount of consistency in the interpretation of these terms is necessary to promote domain interoperability.

The ontology utilizes the complete range of DAML+OIL semantics. The ontology defines numerous is-A relationships as in ‘An `OffensivePlayer` is a subclass of

`FootballPlayer`'. It also supplies additional semantics using domain-specific constraints through the use of specialized DAML+OIL properties and cardinality restrictions. For example, we denote the property `football:fpTeam` as a `daml:UniqueProperty` to suggest that a `FootballPlayer` can have a solitary value for `fpTeam`, that is he plays for to a single team. Some other properties for a `football:FootballPlayer` include `player:name, player:height, football:number, football:yearsPro`, and more. We also enforce the constraint that a `FootballGame` consists of exactly two teams, using the `daml:cardinality` class restriction on the `football:fgTeam` property. This application benefits tremendously from using DAML as the ontology-creation language. It would not have been possible to express such detailed domain knowledge without a powerful semantic language such as DAML.

The Player Hierarchy described by the ontology, is shown in Figure 4.2.

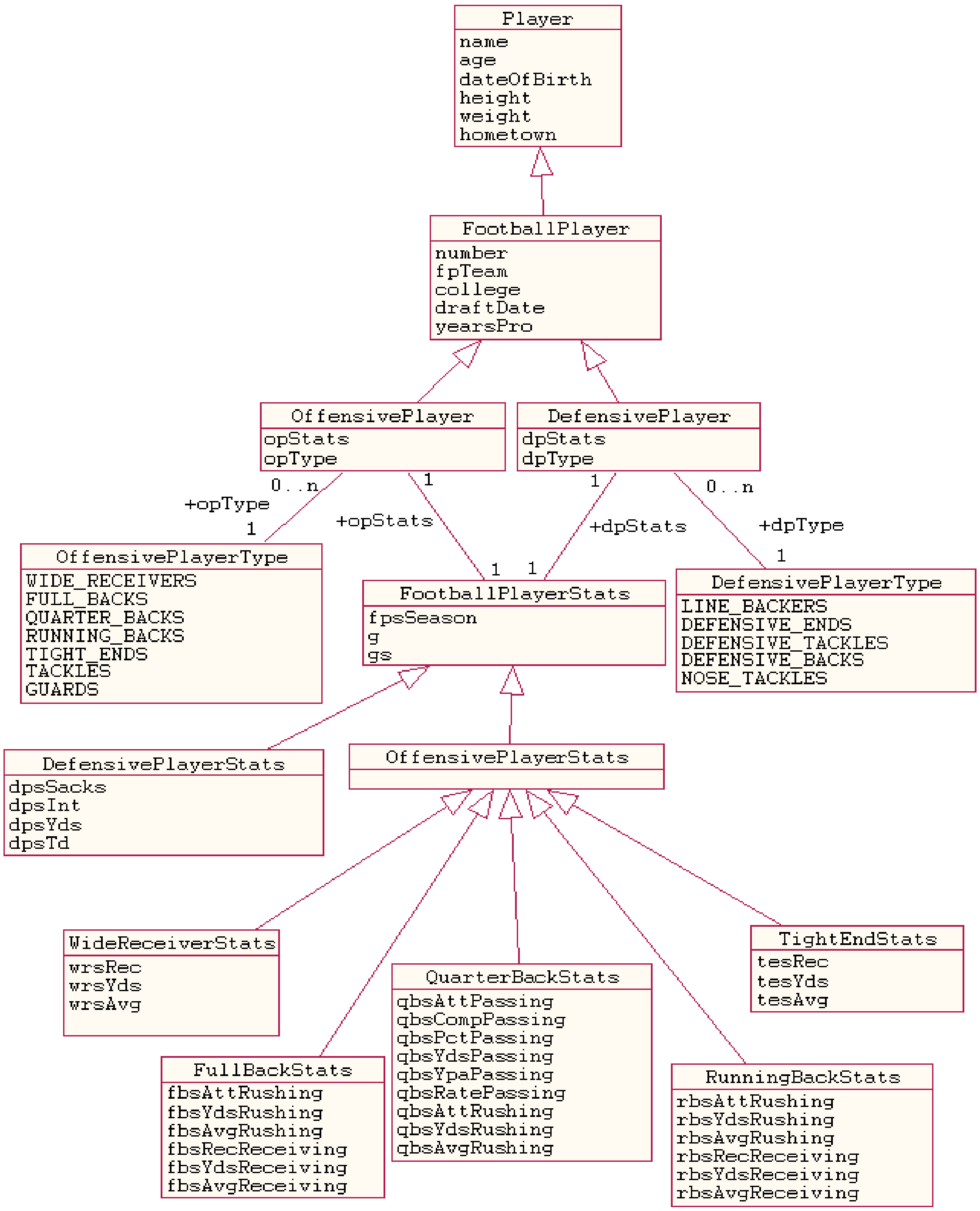


Figure 4.2: Player Hierarchy in the DAML Football Ontology

4.2 Web Scraper Program

We created a simple API for generating DAML markup using the Java programming language. Each entity (class) in the ontology is represented by a separate Java class. Each class contains the properties defined by the ontology, and methods to generate DAML markup. Property restrictions specified by the ontology are encoded into the respective Java classes. The Scraper program retrieves information from web pages (HTML markup), and utilizes the API to create football instances.

A snippet of the DAML instance data is shown in figure 4.3.

```
<football:OffensivePlayer rdf:ID="Albright_Ethan">
 <player:name> <xsd:string rdf:value="Albright_Ethan"/>
</player:name>
 <player:dateOfBirth>
  <dt:date rdf:ID="Albright_Ethan_mm-dd-yy_5-1-1971">
   <dt:Month> <xsd:integer rdf:value="5"/> </dt:Month>
   <dt:Day> <xsd:integer rdf:value="1"/> </dt:Day>
   <dt:Year> <xsd:integer rdf:value="1971"/> </dt:Year>
  </dt:date>
 </player:dateOfBirth>
 <player:age> <xsd:nonNegativeInteger rdf:value="32"/> </player:age>
 <player:height> <xsd:decimal rdf:value="6.5"/> </player:height>
 <player:weight> <xsd:decimal rdf:value="273.0"/> </player:weight>
 <football:number> <xsd:string rdf:value="0"/> </football:number>
 <football:fpTeam>
   <xsd:string rdf:value="Washington_Redskins"/>
 </football:fpTeam>
 <football:college>
   <xsd:string rdf:value="North_Carolina"/>
 </football:college>
 <football:yearsPro>
   <xsd:nonNegativeInteger rdf:value="9"/>
 </football:yearsPro>
 <football:opType>
  <football:OffensivePlayerType>
   <xsd:string rdf:value="center"/>
  </football:OffensivePlayerType>
 </football:opType>
 <football:opStats rdf:ID="Albright_Ethan_Stats">
   <football:OffensivePlayerStats rdf:ID="Albright_Ethan_2003">
     <football:fpsSeason>
       <xsd:nonNegativeInteger rdf:value="2003"/>
     </football:fpsSeason>
     <football:g>
      <xsd:nonNegativeInteger rdf:value="107"/>
     </football:g>
     <football:gs>
      <xsd:nonNegativeInteger rdf:value="0"/>
     </football:gs>
   </football:OffensivePlayerStats>
 </football:opStats>
</football:OffensivePlayer>
```

Figure 4.3: Offensive Player Instance Data in DAML

The Scraper program was developed using the Java Swing HTML Parser. We created an API to generate DAML instance markup for each class in the football ontology. The

Scraper Program uses this API to retrieve football information from the web. The parser works by utilizing callbacks to report important parts in an HTML document.

Each time the parser encounters a start tag, end tag, empty tag, text, or comment it calls the appropriate method from the `javax.swing.text.html.HTMLEditorKit.ParserCallback` class.

The primary task is to create a subclass of `ParserCallback`, and to override the appropriate callback methods to provide our own implementation. We created a subclass of `ParserCallback` to gather player information from specific parts in the document. However, to be able to do this, we needed to know precisely how and where the instance information is laid out on the HTML page.

The Scraper program is a prime example of why the current web is not appropriate for automatic machine processing. It emphasizes the importance of the Semantic Web for machine processing. It is almost impossible for a program to find the relevant information, simply because of the lack of semantics on an HTML web page. Information is described (in a form suitable for human consumption) using special tags, descriptive text, different fonts, or a combination of the above. The structure and content of such pages is not evident to a program browsing this page. Hence, to a large extent, this information needs to be hard-wired into the Scraper callback methods. By responding to these specific points in the document, the Scraper is able to retrieve instance data, and create DAML markup. However, as the web page changes its presentation layout, the

API must be modified to prevent failure. The Semantic Web will help to alleviate these limitations. It would go beyond cosmetics by including tags that also describe what the information content is [25].

The Scraper program has a Graphical User Interface that allows the user to make appropriate player selections. It generates a DAML instance file for the selected player type.

4.3 DAMLJessKB

As noted previously, DAMLJessKB is a reasoning tool that uses Jess as the inference engine. The primary task of DAMLJessKB is to provide an interface for asserting rules and facts into Jess. Jess then uses these rules to carry out basic reasoning. We used Jess version 6.1 p2 for this application.

The basic flow of this library is as follows [10]:

- Read in Jess rules and facts representing the DAML language
- Have RDF API read in the DAML file and create SVO (Subject-Verb-Object) triples
- Take triples and assert into Jess' rete network in VSO form, with some slight escaping of literals and translation
- Have Jess apply the rules of the language to the data
- Apply the agent's rules, queries, etc
- Serialize relevant facts back to DAML

DAMLJessKB uses ARP (Another RDF Parser) to parse RDF/DAML documents. ARP is part of the Jena toolkit [13] developed at Hewlett Packard Labs. ARP generates a stream of RDF triples to be asserted into Jess.

Once the triples are generated, DAMLJessKB rearranges the S-V-O (Subject – Verb – Object) form into V-S-O and adding `PropertyValue` as the head. The word ‘Predicate’ can be used interchangeably with ‘Verb’.Thus another name for the triple V-S-O is P-S-O (Predicate - Subject - Object).

Each triple represents an unordered fact in Jess, and has the format:

```
(PropertyValue <predicate> <subject> <object>)
```

For example:

```
(PropertyValue
       http://www.w3.org/1999/02/22-rdf-syntax-ns#type
       players-in:Brunell_Mark
       football:OffensivePlayer)
```

DAMLJessKB also performs small transformations before asserting facts. Jess doesn't allow names to include tildes so they are escaped. Literals are double quoted as well to avoid being interpreted as multiple terms.

4.3.1 DAML+OIL Rules

DAMLJessKB asserts DAML+OIL semantics as facts into Jess. An example of a rule supplied with the DAMLJessKB package is shown in Figure 4.4.

```
(defrule subclass-instances
  "An instance of a subclass is an instance of the parent class. This
   enforces and makes meaningful the rdfs:subClassOf relationship."
  ;; Note the use of the rdfs predicate instead of the daml version
  (PropertyValue
        http://www.w3.org/2000/01/rdf-schema#subClassOf
        ?child
        ?parent)
  (PropertyValue
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
        ?instance
        ?child)
   =>
  (assert
    (PropertyValue
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
        ?instance
        ?parent)
   )
)
```

Figure 4.4: Rule Implementing DAML+OIL Semantics

These rules are specified using the `defrule` construct in Jess. The following are the rules included with DAMLJessKB [10]:

- An instance of a subclass is an instance of a parent class.
- A subclass of a subclass is a subclass of the root class.
- An instance of a subproperty is an instance of the parent property.
- A subproperty of a subproperty is the subproperty of the root property.
- Two classes can be marked as being the same class and anything said about one will be said about the other.
- Two properties can be similarly marked and anything said about one will be said about the other.

A list of other DAML+OIL semantics specified by DAMLJessKB is shown in Figure 4.5.

- `rdfs:Class is an rdf:Resource`
- `rdf:Property is an rdf:Resource`
- `rdf:type is an rdf:Property`
- `rdf:value is an rdf:Property`
- `rdfs:subClassOf is an rdf:Property`
- `rdfs:Class subclasses rdf:Resource`
- `rdf:Property subclasses rdf:Resource`
- `daml:Class is an rdfs:Class`
- `daml:Class is a subclass of rdfs:Class`
- `daml:sameClassAs is a daml:Property`
- `daml:Property is the same property as rdf:Property`
- `daml:sameClassAs is a subproperty of rdfs:subClassOf`
- `daml:samePropertyAs is a daml:Property`
- `daml:subClassOf is the same property as rdfs:subClassOf`
- `daml:Thing is a daml:Class`
- `daml:Nothing is a daml:Class`

- `daml:Ontology is a daml:Class`

- `daml:ObjectProperty is a rdfs:Class`
- `daml:DatatypeProperty is a rdfs:Class`
- `rdfs:Literal is an rdfs:Class`
- `daml:Literal is the same class as rdfs:Literal`
- `rdfs:comment is a rdf:Property`

- `daml:comment is the same property as rdfs:comment`
- `daml:subPropertyOf is an rdf:Property`
- `daml:subPropertyOf is the same property as rdfs:subPropertyOf`
- `daml:samePropertyAs is a subproperty of rdfs:subPropertyOf`
- `daml:Restriction is an rdfs:Class`
- `daml:Restriction is a subclass of daml:Class`
- `daml:onProperty is an rdf:Property`
- `daml:hasValue is an rdf:Property`
- `daml:imports is an rdf:Property`
- `daml:TransitiveProperty is an rdfs:Class`
- `daml:TransitiveProperty is a subclass of daml:ObjectProperty`

Figure 4.5: List of DAML+OIL Semantics asserted by DAMLJessKB

4.3.2 Domain Rules

In addition to the rules supplied by DAMLJessKB, we developed our own domain-specific rules. These rules make our application more ‘knowledgeable’ about football, and enable it to perform effective reasoning. They may seem obvious to humans with a general knowledge of football, however, they need to be made explicit to a program designed to provide precise information within our domain. We make use of the DAML semantic rules supplied with the DAMLJessKB tool, as well as our own football rules to make our application more 'intelligent'. A complete Jess implementation of these rules is available in the Appendix section.

A list of football rules is shown in figure 4.6.

- A Football Player plays for one team
- A Defensive Player has one type
- An Offensive Player has one type
- A Football Game consists of exactly two teams
- A Football Player's age is always more than his years as a professional
- Two teams cannot have the same mascot
- Two teams cannot have the same website
- A Football Player cannot have negative statistics (i.e. every statistics value must have a minimum value of 0)
- A Past Football Game must not have a begin year later than the current year
- A Future Football Game must not have a begin year earlier than the current year

Figure 4.6: List of Domain Rules

These rules are also constructed using the `defule` construct in Jess. The example shown in Figure 4.7 enforces the rule that a Football Player plays for a single team.

```
;; Rule 1: A FootballPlayer can play for only one team

(defrule one-team
 "A FootballPlayer can play for a single team"
```

```
(PropertyValue
     http://www.w3.org/1999/02/22-rdf-syntax-ns#type
     ?n
     football:FootballPlayer)
?t1 <- (PropertyValue football:fpTeam  ?n        ?a)
?t2 <- (PropertyValue football:fpTeam  ?n        ?b)
?t3 <- (PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            ?a
            http://www.w3.org/2000/10/XMLSchema#string)
?t4 <- (PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            ?b
            http://www.w3.org/2000/10/XMLSchema#string)
?t5 <- (PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            ?a
            ?name1)
?t6 <- (PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            ?b
            ?name2)
(test (> (call ?t2 getFactId) (call ?t1 getFactId)))
(test (> (call ?t4 getFactId) (call ?t3 getFactId)))
(test (> (call ?t6 getFactId) (call ?t5 getFactId)))
=>
(printout WSTDOUT "Violated Rule:A FootballPlayer can play for
                   only one team" crlf)
(printout WSTDOUT "Retracting fact:" (call ?t2 getFactId)  crlf)
(printout WSTDOUT "Retracting fact:" (call ?t4 getFactId) crlf)
(printout WSTDOUT "Retracting fact:" (call ?t6 getFactId) crlf)

(retract ?t2)
```

```
 (retract ?t4)
 (retract ?t6) )
```

Figure 4.7: Domain (Football) Rule Example

DAMLJessKB reads in the DAML instance file shown in Figure 4.3, and generates a stream of triples. The `football:fpTeam` property for a specific `football:OffensivePlayer` is mapped into the set of triples shown in Figure 4.8.

```
(PropertyValue
     http://www.w3.org/1999/02/22-rdf-syntax-ns#type
     players-in:Beverly_Eric
     football:OffensivePlayer)
(PropertyValue
     football:fpTeam
     players-in:Beverly_Eric
     players-in:anonARP379)
(PropertyValue
     http://www.w3.org/1999/02/22-rdf-syntax-ns#type
     players-in:anonARP379
     http://www.w3.org/2000/10/XMLSchema#string)
(PropertyValue
     http://www.w3.org/1999/02/22-rdf-syntax-ns#value
     players-in:anonARP379
     "Detroit_Lions")
```

Figure 4.8: Triples representing the football:fpTeam property

The `one-team` rule works as follows. It tries to find a set of patterns matching the triples shown in Figure 4.8. If it finds more than one such set, we know that the knowledge base contains facts about a football player playing for 2 or more different teams. Every fact asserted into Jess is assigned a unique fact id. Older facts have lower values of fact id. The rule action compares their fact id values, and retracts (removes) facts with the higher fact Id (more recent facts) from the knowledge base.

The original facts (with lower fact Id) are asserted when DAMLJessKB parses DAML instance data generated by the Scraper program. The rule ensures that any new facts asserted do not violate the `one-team` rule for any `football:FootballPlayer`. Any change to the player information should be made directly in the DAML instance file read by the application.

4.4 Jena Semantic Web Toolkit

The Semantic Web Search system accepts user queries by means of a Graphical User Interface (GUI). On initialization, the GUI loads our default ontology `http://www.cse.sc.edu/~dukle/ontologies/football-ont.daml`, and displays the ontology classes and their respective property values respectively.

We utilized the Jena toolkit (version 1.6.1) [13] for this purpose. The toolkit includes an ARP (Another RDF Parser) to parse RDF documents. It also contains a DAML API with classes to manipulate DAML ontologies and extract information from them.

We employed the following Jena classes to display 'Subject', 'Predicate', and 'ObjectProperty' values:

1. **com.hp.hpl.jena.daml.common.DAMLModelImpl**

This class implements the `com.hp.hpl.jena.daml.DAMLModel` interface. It includes methods to load and store DAML ontology information given a URI (Full name all). We utilize the methods provided by this class to store DAML classes and property information specified by the ontology.

2. **com.hp.hpl.jena.daml.common.DAMLClassImpl**

This class implements the `com.hp.hpl.jena.daml.DAMLClass` interface. It represents the Java implementation of a DAML ontology class. It includes methods to traverse the class hierarchy, retrieve properties of classes, and other methods to test elements in the ontology. This class was extensively used to obtain super class – sub class relationships, and to fetch class properties necessary for initializing the application's user interface.

3. **com.hp.hpl.jena.daml.common.DAMLPropertyImpl**

This class implements the `com.hp.hpl.jena.daml.DAMLProperty` interface. It encapsulates a property in a DAML ontology. It also acts as superclass for more semantically meaningful property classes: datatype properties and object properties.

4.5 Search Queries

The application uses the Jess `defquery` construct to retrieve information from the knowledge base. It is a special kind of rule with no right hand side. When executed, the `defquery` returns a list of all the matches.

We discuss two examples of defqueries. Refer to Section 5.3 for a complete list of queries utilized to test our system.

4.5.1 Query Example 1

Find the list of teams for all football players.

The user makes the following selection using the GUI:

Subject:	`football:FootballPlayer`
Predicate:	`football:fpTeam`
ObjectProperty:	`-none-`

The triples for the `football:fpTeam` value of a specific `football:FootballPlayer` instance are shown in Figure 4.7. We need to find a match for all such patterns appearing in the knowledge base. The GUI selections translate into the following query:

```
(defquery search
    (declare (max-background-rules 100))
    (PropertyValue
         http://www.w3.org/1999/02/22-rdf-syntax-ns#type
         ?n
    football:FootballPlayer)
    (PropertyValue
          football:fpTeam
           ?n
           ?res)
    (PropertyValue
          http://www.w3.org/1999/02/22-rdf-syntax-ns#value
          ?res
          ?s) )
```

The query (after execution) returns a `java.util.Iterator` of all matches found. The details of collecting the query results are covered in the Defqueries section, Section 2.9 of the Jess manual [8]. The user interface displays the results.

4.5.2 Query Example 2

The second example deals with advanced queries. Using simple Jess comparison operators such as ‘=’. ‘<>’, ‘>’, ‘>=’, ‘<’, and ‘<=‘, the user can effectively customize his query to find the most relevant results.

Find all football players with a height no less than 6.0

The user makes the following GUI selections:

Subject: `football:FootballPlayer`

Predicate:	`player:height`
ObjectProperty:	`-none-`
Operator:	>=
Search String:	6.0

These selections translate into the following query:

```
(defquery search
     (declare (max-background-rules 100))
     (PropertyValue
          http://www.w3.org/1999/02/22-rdf-syntax-ns#type
          ?n
    football:FootballPlayer)
     (PropertyValue
          player:height
    ?n
    ?res)
     (PropertyValue
          http://www.w3.org/1999/02/22-rdf-syntax-ns#value
          ?res
          ?s&:(or (integerp ?s) (floatp ?s))&:(>= (float ?s) 6.0))
)
```

The added conditions in the final pattern ensure that the results have a value of at least 6.0. The Jess functions `integerp` and `floatp` return `true` for integer and float values respectively. The numeric value is converted to a float and compared with the user search string. The powerful built-in functions, and advanced pattern-matching capabilities of Jess make it well suited for use in Semantic Web searches.

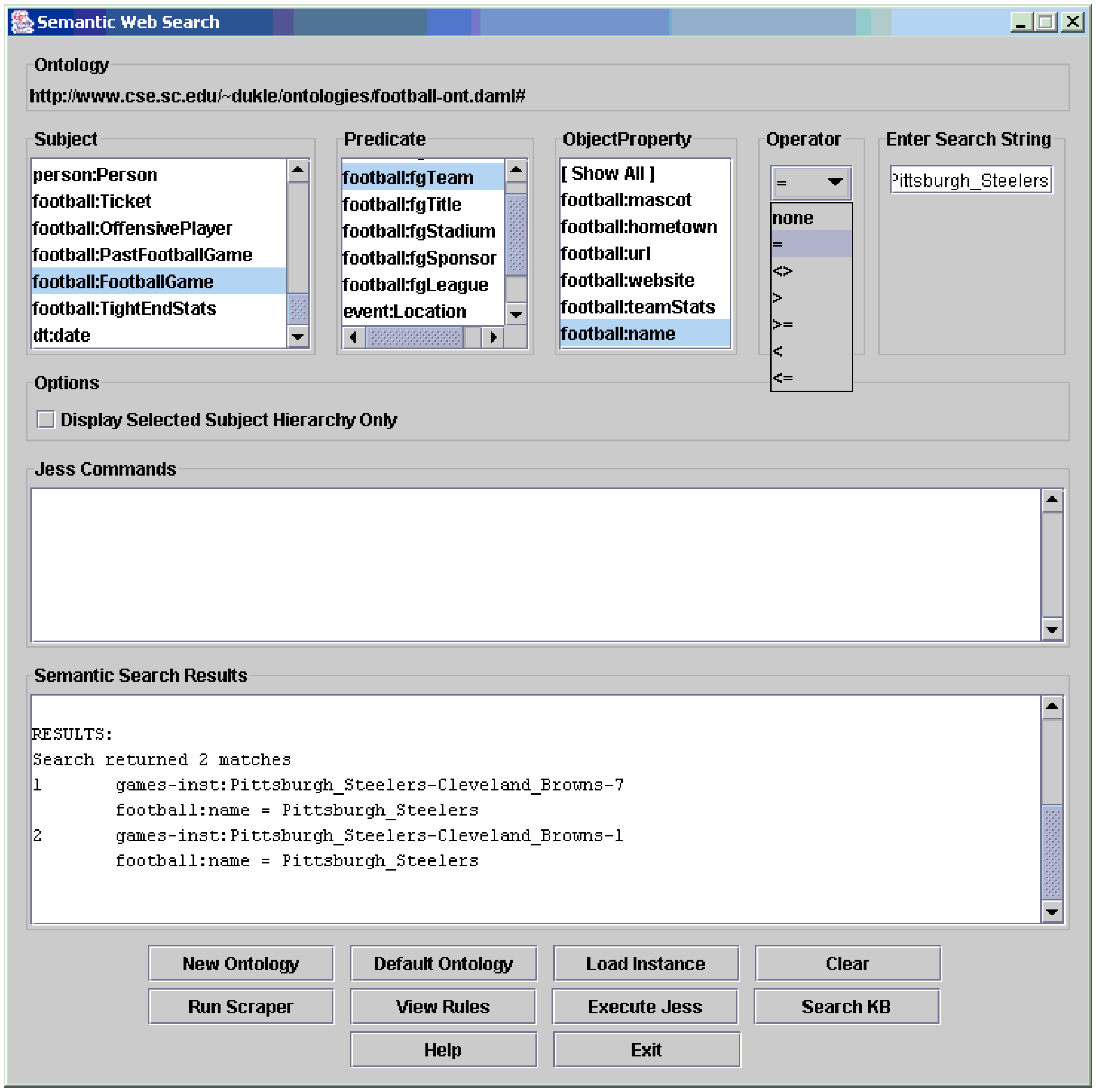
Chapter 5

Tests

5.1 Application User Interface

The user interacts with this system by means of a user interface that allows the user to make appropriate selections for classes (Subject), their properties (Predicate), and properties of Object Properties (if the selected Predicate is an `daml:ObjectProperty`) within the domain. It supports advanced search options by means of comparison operators that help to customize search. In addition, it permits the user to enter additional Jess domain rules, commands and other assertions to test the application's reasoning capabilities.

A snapshot of the Query-Answering (QA) System User Interface is shown in Figure 5.1.



http://www.cse.sc.edu/~dukle/ontologies/football-ont.daml#

Subject: person:Person, football:Ticket, football:OffensivePlayer, football:PastFootballGame, football:FootballGame, football:TightEndStats, dt:date

Predicate: football:fgTeam, football:fgTitle, football:fgStadium, football:fgSponsor, football:fgLeague, event:Location

ObjectProperty: [Show All], football:mascot, football:hometown, football:url, football:website, football:teamStats, football:name

Operator: =, none, =, <>, >, >=, <, <=

Enter Search String: Pittsburgh_Steelers

Display Selected Subject Hierarchy Only

```
RESULTS:
Search returned 2 matches
1       games-inst:Pittsburgh_Steelers-Cleveland_Browns-7
        football:name = Pittsburgh_Steelers
2       games-inst:Pittsburgh_Steelers-Cleveland_Browns-1
        football:name = Pittsburgh_Steelers
```

Figure 5.1: Query-Answering System Graphical User Interface

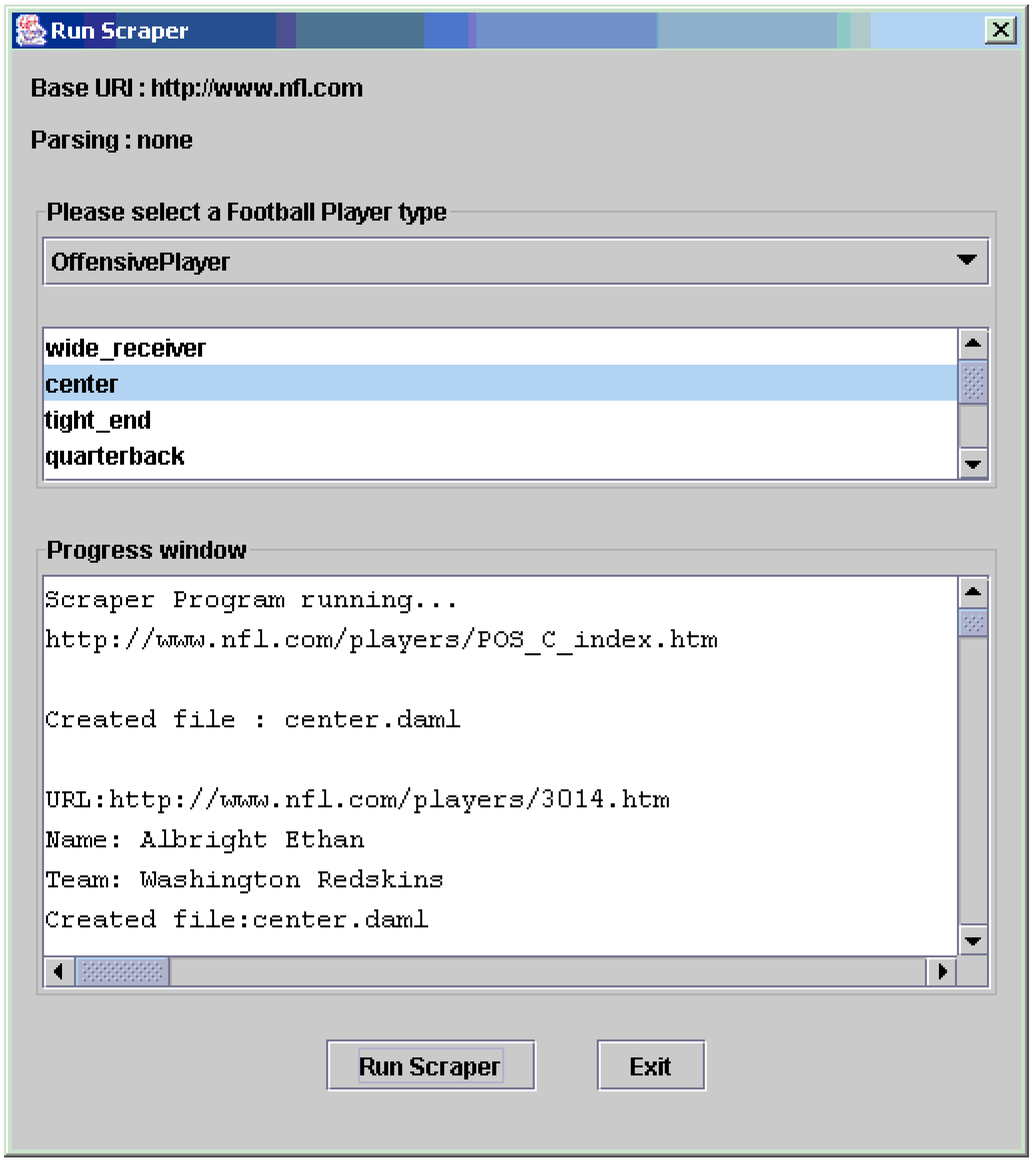
The application loads our default ontology,

http://www.cse.sc.edu/~dukle/ontologies/football-ont.daml

The ontology classes and their respective properties are displayed using the ‘Subject’ and ‘Predicate’ list boxes respectively. The Jena toolkit dynamically reads in the ontology and obtains this class-property information. This ensures that the application is domain-neutral, it would function with any domain ontology and not just our domain – football.

5.2 Testing the Scraper Program

The Scraper program is run to ensure the DAML instance information is kept current. At present, this program only retrieves `FootballPlayer` data, i.e. automated information retrieval of Teams, and FootballGames is not supported. The program is activated using the application’s ‘Run Scraper’ button. The Scraper user interface, shown in Figure 5.2, prompts the user to select the appropriate football player type.



```
Scraper Program running...
http://www.nfl.com/players/POS_C_index.htm

Created file : center.daml

URL:http://www.nfl.com/players/3014.htm
Name: Albright Ethan
Team: Washington Redskins
Created file:center.daml
```

Figure 5.2: Scraper Program Graphical User Interface

The ‘Progress Window’ lets the user keep track of the process. A DAML instance file with the selected player information is generated at the end of the process. For example, if the user selects an offensive player type with the value ‘center’, a `center.daml` file is created. A snippet of this DAML file was shown in Figure 4.3.

5.3 Testing the QA system

The application requires at least one DAML instance file to provide search capabilities. A URI to the DAML file can be loaded using the ‘Load Instance’ button. The ‘Search KB’ button runs the search. We tested our application using three instance files containing DAML data for football players, teams and games.

Instance files:

1. `http://www.cse.sc.edu/~dukle/ontologies/players-inst.daml`
2. `http://www.cse.sc.edu/~dukle/ontologies/teams-inst.daml`
3. `http://www.cse.sc.edu/~dukle/ontologies/games-inst.daml`

In response to the user selection, the interface automatically creates a Jess `defquery` encapsulating the search information. Jess executes the query, searching for the required pattern in its knowledge base, and sends results back to the interface. Results are displayed in the ‘Semantic Search Results’ area.

5.3.1 Test Query 1

Find the names of all Offensive players

User Interface Selection:

Subject:	`football:OffensivePlayer`
Predicate:	`player:name`
ObjectProperty:	`-none-`
Operator:	`-none-`
Enter Search String:	`-none-`

Jess Query:

```
(defquery search
    (declare (max-background-rules 100))
    (PropertyValue
          http://www.w3.org/1999/02/22-rdf-syntax-ns#type
          ?n
          football:OffensivePlayer)
    (PropertyValue
          player:name
          ?n
          ?res)
    (PropertyValue
          http://www.w3.org/1999/02/22-rdf-syntax-ns#value
          ?res
         ?s) )
```

The search returns a list of `player:name` of all `football:OffensivePlayer` found in the knowledge base.

Results:

```
Search returned 16 matches

1     players-in:Ackerman_Tom
      player:name = Ackerman_Tom
2     players-in:Anelli_Mark
      player:name = Anelli_Mark
3     players-in:Anderson_Scotty
      player:name = Anderson_Scotty
4     players-in:Beverly_Eric
      player:name = Beverly_Eric
5     players-in:Brunell_Mark
      player:name = Brunell_Mark
6     players-in:Alstott_Mike
      player:name = Alstott_Mike
7     players-in:Adams_Flozell
      player:name = Adams_Flozell
8     players-in:Alexander_Stephen
      player:name = Alexander_Stephen
9     players-in:Alexander_Derrick
      player:name = Alexander_Derrick
10    players-in:Abdullah_Rabih
      player:name = Abdullah_Rabih
11    players-in:Anderson_Bennie
      player:name = Anderson_Bennie
12    players-in:Barnes_Darian
      player:name = Barnes_Darian
13    players-in:Benjamin_Ryan
      player:name = Benjamin_Ryan
14    players-in:Brady_Tom
      player:name = Brady_Tom
15    players-in:Ashworth_Tom
      player:name = Ashworth_Tom
```

```
16    players-in:Alexander_Shaun
      player:name = Alexander_Shaun
```

5.3.2 Test Query 2

Find names of all football players

User Interface Selection:

Subject:	`football:FootballPlayer`
Predicate:	`player:name`
ObjectProperty:	`-none-`
Operator:	`-none-`
Enter Search String:	`-none-`

Jess Query:

```
(defquery search
    (declare (max-background-rules 100))
    (PropertyValue
         http://www.w3.org/1999/02/22-rdf-syntax-ns#type
         ?n
         football:FootballPlayer)
    (PropertyValue
         player:name
         ?n
         ?res)
    (PropertyValue
         http://www.w3.org/1999/02/22-rdf-syntax-ns#value
         ?res
         ?s)
)
```

Results:

```
Search returned 26 matches

1     players-in:Almanzar_Luis
      player:name = Almanzar_Luis
2     players-in:Abraham_Donnie
      player:name = Abraham_Donnie
3     players-in:Armstead_Jessie
      player:name = Armstead_Jessie
4     players-in:Ashworth_Tom
      player:name = Ashworth_Tom
5     players-in:Alstott_Mike
      player:name = Alstott_Mike
6     players-in:Allen_Will
      player:name = Allen_Will
7     players-in:Ackerman_Tom
      player:name = Ackerman_Tom
8     players-in:Brunell_Mark
      player:name = Brunell_Mark
9     players-in:Alexander_Derrick
      player:name = Alexander_Derrick
10    players-in:Alexander_Shaun
      player:name = Alexander_Shaun
11    players-in:Beverly_Eric
      player:name = Beverly_Eric
```

```
12    players-in:Clancy_Kendrick
      player:name = Clancy_Kendrick
13    players-in:Abraham_John
      player:name = Abraham_John
14    players-in:Anderson_Scotty
      player:name = Anderson_Scotty
15    players-in:Alexander_Stephen
      player:name = Alexander_Stephen
16    players-in:Ferguson_Jason
      player:name = Ferguson_Jason
17    players-in:Anderson_Bennie
      player:name = Anderson_Bennie
18    players-in:Armstrong_Trace
      player:name = Armstrong_Trace
19    players-in:Benjamin_Ryan
      player:name = Benjamin_Ryan
20    players-in:Barnes_Darian
      player:name = Barnes_Darian
21    players-in:Brady_Tom
      player:name = Brady_Tom
22    players-in:Anelli_Mark
      player:name = Anelli_Mark
23    players-in:Abdullah_Rabih
      player:name = Abdullah_Rabih
24    players-in:Adams_Sam
      player:name = Adams_Sam
25    players-in:Adams_Flozell
      player:name = Adams_Flozell
26    players-in:Adams_Keith
      player:name = Adams_Keith
```

In our ontology, we have defined `football:FootballPlayer` to be a super class of both `football:OffensivePlayer`, and `football:DefensivePlayer`. We have also asserted the rule "An instance of a subclass is an instance of the parent class" using the DAML semantics file supplied with DAMLJessKB. Jess executes this rule, and asserts each instance of a subclass as an instance of the parent class using a separate fact. This rule ensures that a search for a predicate of a parent class returns values of the predicate for all its subclasses. The search in 'Test Query 1' returned 16 names of type `football:OffensivePlayer`, while the current search returned 26 results. We verified that the remaining 10 names not listed in 'Test Query 1' were indeed of type `football:DefensivePlayer`. This search demonstrates the inferencing capabilities of our application, based on rules specifying the semantics.

User Selection:

Subject: `football:DefensivePlayer`

Predicate: `player:name`

ObjectProperty: `-none-`

Operator: `-none-`

Enter Search String: `-none-`

Results:

```
Search returned 10 matches
1     players-in:Ferguson_Jason
      player:name = Ferguson_Jason
```

```
2     players-in:Adams_Keith
      player:name = Adams_Keith
3     players-in:Armstrong_Trace
      player:name = Armstrong_Trace
4     players-in:Abraham_Donnie
      player:name = Abraham_Donnie
5     players-in:Clancy_Kendrick
      player:name = Clancy_Kendrick
6     players-in:Almanzar_Luis
      player:name = Almanzar_Luis
7     players-in:Abraham_John
      player:name = Abraham_John
8     players-in:Armstead_Jessie
      player:name = Armstead_Jessie
9     players-in:Adams_Sam
      player:name = Adams_Sam
10    players-in:Allen_Will
      player:name = Allen_Will
```

5.3.3 Test Query 3

Find all football games played at 3COM Park stadium.

This search demonstrates the advanced search feature. The search results can be filtered using a comparison operator and a string specifying the match pattern.

User Interface Selection:

Subject:	`football:FootballGame`
Predicate:	`football:fgStadium`
ObjectProperty:	`-none-`
Operator:	=
Enter Search String:	`3COM_Park`

Jess Query:

```
(defquery search
    (declare (max-background-rules 100))
    (PropertyValue
         http://www.w3.org/1999/02/22-rdf-syntax-ns#type
         ?n
         football:FootballGame)
    (PropertyValue
         football:fgStadium
         ?n
         ?res)
    (PropertyValue
         http://www.w3.org/1999/02/22-rdf-syntax-ns#value
         ?res
         ?s&:(= (str-compare ?s "3COM_Park") 0))
)
```

Results:

```
Search returned 2 matches

1     games-inst:New_York_Giants-San_Francisco_49ers-4
      football:fgStadium = 3COM_Park
2      games-inst:San_Francisco_49ers-Cleveland_Browns-5
      football:fgStadium = 3COM_Park
```

5.3.4 Test Query 4

Find all football players with a weight no less that 250

This test demonstrates advanced search with numeric values.

User Interface Selection:

Subject:	`football:FootballPlayer`
Predicate:	`player:weight`
ObjectProperty:	`-none-`
Operator:	>=
Enter Search String:	`250`

Jess Query:

```
(defquery search
    (declare (max-background-rules 100))
    (PropertyValue
       http://www.w3.org/1999/02/22-rdf-syntax-ns#type
       ?n
       football:FootballPlayer)
    (PropertyValue
       player:weight
       ?n
       ?res)
    (PropertyValue
       http://www.w3.org/1999/02/22-rdf-syntax-ns#value
       ?res
       ?s&:(or (integerp ?s) (floatp ?s))&:(>= (float ?s) 250)))
```

Results:

```
Search returned 14 matches

1      players-in:Almanzar_Luis
       player:weight = 295.0
2      players-in:Ashworth_Tom
       player:weight = 305.0
3      players-in:Ackerman_Tom
       player:weight = 292.0
4      players-in:Beverly_Eric
       player:weight = 300.0
5      players-in:Clancy_Kendrick
       player:weight = 289.0
```

```
6     players-in:Abraham_John
      player:weight = 256.0
7     players-in:Alexander_Stephen
      player:weight = 250.0
8     players-in:Ferguson_Jason
      player:weight = 305.0
9     players-in:Anderson_Bennie
      player:weight = 335.0
10    players-in:Armstrong_Trace
      player:weight = 275.0
11    players-in:Barnes_Darian
      player:weight = 250.0
12    players-in:Anelli_Mark
      player:weight = 265.0
13    players-in:Adams_Sam
      player:weight = 330.0
14    players-in:Adams_Flozell
      player:weight = 357.0
```

5.3.5 Test Query 5

Find all defensive players born after the year 1975

This test query demonstrates querying of values for object properties.

The `player:dateOfBirth` is a DAML Object Property that has range `dt:Date`. `dt:Date` has properties `dt:Month`, `dt:Day`, `dt:Year`, and `dt:DayOfWeek`. `dt` is a namespace prefix that refers to UMBC [19] Calendar ontology. We wish to retrieve information by querying on the `dt:Year` property of the `player:dateOfBirth` Object Property.

User Interface Selection:

Subject:	`football:DefensivePlayer`
Predicate:	`player:dateOfBirth`
ObjectProperty:	`dt:Year`
Operator:	>
Enter Search String:	`1975`

Jess Query:

```
(defquery search
    (declare (max-background-rules 100))
    (PropertyValue
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
        ?n
        football:DefensivePlayer)
    (PropertyValue
        player:dateOfBirth
        ?n
        ?res)
    (PropertyValue
        dt:Year
        ?res
        ?y)
    (PropertyValue
         http://www.w3.org/1999/02/22-rdf-syntax-ns#value
         ?y
         ?s&:(or (integerp ?s) (floatp ?s))&:(> (float ?s) 1975))
)
```

Results:

```
Search returned 5 matches

1     players-in:Almanzar_Luis
      dt:Year = 1976
2     players-in:Abraham_John
      dt:Year = 1978
3     players-in:Allen_Will
      dt:Year = 1978
4     players-in:Adams_Keith
      dt:Year = 1979
5     players-in:Clancy_Kendrick
      dt:Year = 1978
```

5.3.6 Test Query 6: Working with different domains

We tested our application to ensure it provides query-answering capabilities for any domain – not just football. We used the Courses ontology [33] created at Robotics Institute at Carnegie Mellon University. The Courses ontology provides description about courses at a university, specific course information, and information about instructors teaching those courses. We created a DAML instance file describing courses offered at the University of South Carolina in the Fall 2003 semester.

The first step is to load the 'Courses' ontology using the 'New Ontology' button.
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-courses-ont.daml

Next, we load the instance file using the application's 'Load Instance' button.
http://www.cse.sc.edu/~dukle/ontologies/cmu-inst.daml

Query:

Find the email addresses of all instructors teaching courses.

User Interface Selection:

Subject:	`cmu-ri-cou:Course`
Predicate:	`cmu-ri-cou:hasInstructor`
ObjectProperty:	`cmu-ri-peo:hasEmail`
Operator:	`-none-`
Enter Search String:	`-none-`

Jess Query:

```
(defquery search
   (declare (max-background-rules 100))
   (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

```
          ?n
          cmu-ri-cou:Course)
   (PropertyValue
          cmu-ri-cou:hasInstructor
          ?n
          ?res)
   (PropertyValue
          cmu-ri-peo:hasEmail
          ?res
          ?y)
   (PropertyValue
          http://www.w3.org/1999/02/22-rdf-syntax-ns#value
          ?y
          ?s)
)
```

Results:

```
Search returned 8 matches

1     cmu-inst:CSCE531_Compiler_Construction
      cmu-ri-peo:hasEmail = fenner@cse.sc.edu
2     cmu-inst:CSCE782_Multiagent_Systems
      cmu-ri-peo:hasEmail = vidal@sc.edu
3     cmu-inst:CSCE513_Computer_Architecture
      cmu-ri-peo:hasEmail = kcameron@cse.sc.edu
4     cmu-inst:CSCE565_Computer_Graphics
      cmu-ri-peo:hasEmail = songwang@cse.sc.edu
5     cmu-inst:CSCE520_Database_System_Design
      cmu-ri-peo:hasEmail = Unknown
```

```
6     cmu-inst:CSCE582_Bayesian_Network_Graphs
      cmu-ri-peo:hasEmail = mgv@cse.sc.edu
7     cmu-inst:CSCE522_Info_Security_Principles
      cmu-ri-peo:hasEmail = farkas@cse.sc.edu
8     cmu-inst:CSCE742_Software_Architectures
      cmu-ri-peo:hasEmail = matthews@cse.sc.edu
```

5.4 Testing our Domain Rules

As noted previously in section 4.3.2, we designed a set of rules for our domain. We tested our system using these rules to ensure it makes correct, and consistent inferences within our domain. We try to assert facts that violate these rules, and then check if these new facts are identified and retracted from the knowledge base.

We entered these Jess rules, and assertions in the ‘Jess Commands’ text area, and use the ‘Execute Jess’ button to run these Jess commands. After executing the rule, we used invalid assertions to test the system.

5.4.1 Test Rule 1

A Football Player plays for one team.

Using our search system, we observe that `players-in:Adams_Keith` has a `football:fpTeam` value `Philadelphia_Eagles`.

We now try to add facts stating that `players-in:Adams_Keith` plays for two other teams – `Chicago_Bears` and `Dallas_Cowboys`.

```
(assert(PropertyValue
            football:fpTeam
            players-in:Adams_Keith
            players-in:_a1))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            players-in:_a1
            http://www.w3.org/2000/10/XMLSchema#string))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            players-in:_a1
            "Chicago_Bears"))
(assert(PropertyValue
            football:fpTeam
            players-in:Adams_Keith
            players-in:_a2))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            players-in:_a2
            http://www.w3.org/2000/10/XMLSchema#string))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            players-in:_a2
            "Dallas_Cowboys"))
```

We assert these facts into Jess, and observe that the total number of facts increases as a result of these assertions.

```
Semantic Search Results:

--- List Facts
Total of 6174 facts!
Total facts before execute command: 6174

--- List Facts
Total of 6184 facts!
Total facts after execute command: 6184
```

The number of facts increases by 10 as a result of these assertions. A thing to note here is that the fact count does not necessarily increase by the exact number of assertions (6 in this case). This is because additional facts are automatically inferred, and asserted into Jess as a result of the semantic rules provided with DAMLJessKB.

We query the knowledge base for values of `football:fpTeam` for all `football:FootballPlayer`, and find that `players-in:Adams_Keith` has three values for `football:fpTeam` (i.e. he plays for 3 teams)

Results (Relevant values included):

```
Semantic Search Results:

26    players-in:Adams_Keith
      football:fpTeam = Philadelphia_Eagles
27    players-in:Adams_Keith
      football:fpTeam = Chicago_Bears
28    players-in:Adams_Keith
      football:fpTeam = Dallas_Cowboys
```

Our domain rules suggest that a football player can play for a single team. This is clearly a domain-rule violation. We now execute the Jess rule – `'A FootballPlayer plays for a single team'`, to make Jess aware of such restrictions.

```
;; Rule 1: A FootballPlayer can play for only one team

(defrule one-team
 "A FootballPlayer can play for a single team"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:FootballPlayer)
 ?t1 <- (PropertyValue football:fpTeam  ?n        ?a)
 ?t2 <- (PropertyValue football:fpTeam  ?n        ?b)
 ?t3 <- (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#type
              ?a
              http://www.w3.org/2000/10/XMLSchema#string)
 ?t4 <- (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#type
              ?b
              http://www.w3.org/2000/10/XMLSchema#string)
 ?t5 <- (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#value
              ?a
              ?name1)
 ?t6 <- (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#value
              ?b
              ?name2)
 (test (> (call ?t2 getFactId) (call ?t1 getFactId)))
 (test (> (call ?t4 getFactId) (call ?t3 getFactId)))
 (test (> (call ?t6 getFactId) (call ?t5 getFactId)))
 =>
 (printout WSTDOUT "Violated Rule:A FootballPlayer can play for
                    only one team" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t2 getFactId) crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t4 getFactId) crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t6 getFactId) crlf)
 (retract ?t2)
 (retract ?t4)
 (retract ?t6) )
```

The `one-team` rule was described previously in Section 4.3.2. On execution, Jess looks for newer patterns violating the rule, and retracts them from the knowledge base.

```
Semantic Search Results:

Violated Rule:A FootballPlayer can play for only one team
players-in:Adams_Keith plays for Dallas_Cowboys
Retracting fact:6182
Retracting fact:6183
Retracting fact:6184
Violated Rule:A FootballPlayer can play for only one team
players-in:Adams_Keith plays for Chicago_Bears
Retracting fact:6179
Retracting fact:6180
Retracting fact:6181

--- List Facts
Total of 6184 facts!
Total facts before execute command: 6184

--- List Facts
Total of 6178 facts!
Total facts after execute command: 6178
```

We repeat our search for values of `football:fpTeam` for all `football:FootballPlayer`.

```
Semantic Search Results:

26    players-in:Adams_Keith
      football:fpTeam = Philadelphia_Eagles
```

We note that the original fact (present in the DAML instance file) that `players-in:Adams_Keith` plays for `football:fpTeam` named `Philadelphia_Eagles` was retained. All other illegal assertions were retracted.

We followed the same procedure to test other domain rules. We fed rules into the inference engine, and verified that illegal facts or assertions were removed from the knowledge base.

From the test cases, we find that our application is able to answer simple as well as advanced domain-specific queries. It is capable of support reasoning based on DAML+OIL semantics as evident from ‘Test Query 2’. In addition, our application utilizes the football rules to ensure the integrity of the knowledge base. The application acquires the ‘intelligence’ and reasoning capabilities required to correctly answer queries by utilizing the semantics specified by DAML+OIL and domain-specific rules.

CHAPTER 6

Conclusion

6.1 Analysis

A lot of challenges must be overcome in order to realize the Semantic Web vision. The first problem lies in utilizing effective semantic markup to represent the vast amount of data on the web. Ontologies will be necessary to provide meaning and definitions of the content. We must also be able to integrate the data obtained from diverse sources. Instead of using a single large complex ontology for data representation, we must be able to create small ontologies that support easy integration with other ontologies on the web. Finally, the proposed system must scale to the enormous size of the World Wide Web.

We have described a Semantic Web query-answering system that facilitates information retrieval within our domain (football) with some degree of reasoning. Currently, the information describing football is encoded using the Hyper Text Markup Language (HTML). We developed a Scraper program for automated information retrieval from these web pages. Specifically, we chose the NFL website (http://www.nfl.com) because of its relevance to our domain. We created a Java API to generate DAML instance markup for each class in the football ontology. The Scraper Program uses this

API to create DAML instances. This information was used by our query-answering system to answer domain-specific queries. The Scraper program was a prime example of how the current web focuses on information visualization, but does not provide semantics for software programs to find or interpret it. Information is described (in a form suitable for human consumption) using special tags, descriptive text, different fonts, or a combination of the above. The structure and content of such pages is not evident to a program browsing this page. When a web page changes its presentation layout, the API must be modified to prevent failure. Due to this structural dependency, our Scraper Program does not support reusability.

The basis of our system is the set of ontologies used to define entities in our domain, and DAMLJessKB, the reasoning tool. Ontologies are described using DAML+OIL. We chose DAML+OIL because it is a semantically rich language that provides machines with capability to read, interpret, and perform inferencing over the data. In addition to the definitions provided by RDF/S, DAML+OIL provides rules for describing constraints such as cardinality, domain and range restrictions, and relationships among resources. Our application supports ontology reuse essential for effective data integration. We link to pre-existing ontologies to add semantics to our ontologies. This promotes common understanding of concepts, and avoids redefinition of terms defined in other ontologies. Ontology sharing is an important aspect of the Semantic Web because of the heterogeneity among domains on the web. Some amount of consistency in the interpretation of these terms is necessary to promote domain interoperability. Ontologies are also used to perform basic consistency checking of the data – to check whether the

DAML instance data used by our search system conforms to the ontology specifications. However, it will be a difficult task to convince the (web page) authors to use DAML to annotate their web pages. They would be willing to do if and only if this additional markup brings about a significant change in information retrieval – it provides more precise results than the ones obtained from the current setup.

We used DAMLJessKB as our reasoning tool. DAMLJessKB was mainly concerned with providing an interface to load DAML ontologies and instances, and to supply DAML+OIL semantics and domain-specific semantics to Jess. The inference engine was used to answer implicit and explicit queries, and perform reasoning based on knowledge represented in DAML. The fact that Jess uses the efficient Rete algorithm for its operations also helps improve performance while searching patterns in response to queries. Rete keeps the computational complexity linear in the size of the knowledge base. DAMLJessKB supplies the basic facts and rules that facilitate inferencing on hierarchical relationships (subclass and subproperty). In addition, we provided relevant rules for reasoning over instance values and concepts defined in our ontology. We also incorporated the use of Jess operators and pre-defined functions to efficiently filter search results in response to user queries.

The combination of domain-specific rules, DAML+OIL semantics, and Jess functionality provided us with an effective query-answering service with reasoning capabilities. We demonstrated how semantic markup can be used to create ontologies and annotate web content. In Section 5.3, we tested our system with different user inputs to demonstrate the precise results obtained from the use of DAML+OIL markup. Although this system was primarily tested within the 'football' domain, it can be effectively adapted for answering queries in other domains utilizing semantic markup. We advocate the qualities of DAML+OIL as a semantic markup language, and believe that the use of DAML+OIL markup to represent domain knowledge will promote automated reasoning and precise information retrieval.

6.2 Future Work

A limitation of this system is that although it is domain-independent, the system is still restricted to answering queries pertaining to that particular domain or ontology. It does not provide the capability to answer queries relating two or more disjoint domains. It operates on individual domains at a given time, contrary to web search engines that operate on a global context. The future work should look at applying these search techniques to a wider range of domains. In addition, to facilitate precise searches, semantic markup can be used as indexing terms for web documents. Research in this area is described in [28].

The future work should also explore the possibility of extending the capabilities of this application into an agent-based system. The agent for our domain will be made responsible for providing these query services. DAML-S can be considered as a markup tool for advertising these services. DAML-S is built on top of DAML+OIL, and it supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. This will result in a reasonable amount of automation, and promote inter-agent communication. Other agents will be able to interpret the DAML-S markup to determine the queries supported by this system.

Inter-agent communication can be made possible by utilizing the FIPA (Foundation for Intelligent Physical Agents) or JADE (Java Agent DEvelopment framework) standards for agent development. The main responsibility of these standards is specifying and maintaining an agent communication language, along with libraries of predefined communicative acts, interaction protocols and content languages. Creating agents that understand and support DAML will be a huge step closer to fulfilling the Semantic Web.

Another direction for future growth would be to make the inference-testing capabilities independent of the Jess scripting language. This application, at present, relies on the user entering Jess code (in the form of assertions) to test DAML semantics as well as domain rules. We could allow the possibility for the (authorized) user to enter additional instance data or change existing data through the user interface. The system should automatically convert this into a Jess assertion, and depending on whether the domain rule is violated either assert or retract the fact from the knowledge base.

REFERENCES

[1] Tim Berners-Lee, James Hendler and Ora Lasilla, "The Semantic Web", The Scientific American, May 2001.

[2] James Hendler, "Agents and the Semantic Web", IEEE Intelligent Systems, 2001.

[3] Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks, "The Semantic Web: The Roles of XML and RDF", IEEE Internet Computing, October 2000.

[4] Michel Klein, "XML, RDF and relatives", IEEE Intelligent Systems, 2001.

[5] World Wide Web Consortium, Namespaces in XML, January 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114/

[6] Adam Pease, Why Use DAML?, April 2002. http://www.daml.org/2002/04/why.html

[7] Youyong Zou, Tim Finin, Yun Peng, Anupam Joshi, and Scott Cost, "Agent Communication in DAML World", 2001.

[8] Ernest Friedman-Hill, Jess, the rule engine for the Java platform, 2000. http://herzberg.ca.sandia.gov/jess/

[9] Ernest Friedman-Hill, The Rete Algorithm, 2000. http://herzberg.ca.sandia.gov/jess/docs/52/rete.html

[10] Joe Kopena, DAMLJessKB. http://plan.mcs.drexel.edu/projects/legorobots/design/software/DAMLJessKB/

[11] Joe Kopena and William C. Regli, "DAMLJessKB: A Tool for Reasoning with the SemanticWeb", October 2002.

[12] DARPA, DAML (DARPA Agent Markup Language), 2001. http://www.daml.org

[13] Hewlett Packard Labs, Jena Semantic Web Toolkit. http://www.hpl.hp.com/semweb/jena.htm

[14] National Football League.
http://www.nfl.com

[15] Elliott Rusty Harold, Java Network Programming, CA, O'Reilly and Associates, 2000.

[16] Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Ian Soboroff, Harry Chen, Lalana Kagal, Filip Perich, Youyong Zou, and Sovrin Tolia, "Ittalks: A Case Study in the Semantic Web and DAML+OIL", IEEE Intelligent Systems, 2002.

[17] SWI-Prolog
http://www.swi-prolog.org/

[18] XSB
http://xsb.sourceforge.net

[19] University of Maryland, Baltimore County, DAML at UMBC.
http://daml.umbc.edu/reasoning

[20] Stanford University, Knowledge Systems Laboratory.
http://www.ksl.stanford.edu/projects/DAML/

[21] Deborah L. McGuinness, "Ontologies Come of Age", Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential, MIT Press, 2002.

[22] James Hendler and Deborah McGuinness, "The DARPA Agent Markup Language", IEEE Intelligent Systems, 15(6):72–73, November/December 2000.

[23] Talk Ontology by University of Maryland, Baltimore County
http://daml.umbc.edu/ontologies/talk-ont.daml

[24] Alexander Maedche and Steffen Staab, "Ontology Learning for the Semantic Web", IEEE Intelligent Systems, 2001.

[25] Mark Frauenfelder, "A Smarter Web", Technology Review, November 2001.

[26] Ora Lassila, and S. R. R. (eds), Resource Description Framework (rdf) model and syntax specification. W3C Recommendation, February 1999.
http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[27] Tim Berners-Lee, Semantic Web on XML, XML 2000 Washington DC, December 2000.
http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html

[28] Tim Finin, Anupam Joshi, R. Scott Cost, James Mayfield, and Urvi Shah, "Information Retrieval on the Semantic Web", ACM Conference on Information and Knowledge Management , November 2002.

[29] Lockheed Martin, "AeroText Products: Building a Customizable Information Extraction System", April 2003.
https://mds.external.lmco.com/products/gims/aero/wpaper.html

[30] James Mayfield, Paul McNamee, and Christine Piatko, "The JHU/APL HAIRCUT System at TREC-8", 2001.

[31] Google Search Engine
http://www.google.com

[32] K. Yue, "The Semantic Web, HTML and XML: An Example", January 2002.
http://dcm.cl.uh.edu/yue/papers/SemanticWebandHTML.asp

[33] Courses Ontology by Carnegie Mellon University
http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-courses-ont.daml

APPENDIX

DAML Player Ontology

```
            <!-- Player Ontology -->
      <!-- by Kapil Dukle (kapilrd@yahoo.com) -->

<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:daml ="http://www.daml.org/2001/03/daml+oil#"
 xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
 xmlns:dt ="http://daml.umbc.edu/ontologies/calendar-ont#"
 xmlns =
   "http://www.cse.sc.edu/~dukle/ontologies/player-ont.daml#"
>

<daml:Ontology about="">
 <daml:versionInfo>
  $player-ont.daml, v 1.4, 2001/06/24 04:35:49, Kapil Dukle$
 </daml:versionInfo>
 <rdfs:comment>A Basic Player Ontology.</rdfs:comment>
 <daml:imports
  rdf:resource="http://www.daml.org/2001/03/daml+oil" />
 <daml:imports
  rdf:resource="http://daml.umbc.edu/ontologies/calendar-ont" />
</daml:Ontology>

            <!-- Basic Classes -->

<daml:Class rdf:ID="Player">
 <rdfs:label>Player</rdfs:label>
 <rdfs:comment> A Player defining an individual in any sport
 </rdfs:comment>
</daml:Class>

            <!-- Properties for Player class -->

<daml:DatatypeProperty rdf:ID="name">
 <rdfs:comment>
  Player can have one or more names. eg. Official name,
  Nickname.
 </rdfs:comment>
 <rdfs:domain rdf:resource="#Player" />
 <rdfs:range
   rdf:resource="http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>
```

```
<daml:ObjectProperty rdf:ID="dateOfBirth">
 <rdfs:comment> dateOfBirth is a UniqueProperty (can have only
  one date of birth). </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#Player" />
 <rdfs:range rdf:resource=
          "http://daml.umbc.edu/ontologies/calendar-ont#date" />
</daml:ObjectProperty>

<daml:DatatypeProperty rdf:ID="age">
 <rdfs:comment> age is a UniqueProperty (can have only one age).
</rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#Player" />
 <rdfs:range rdf:resource
    ="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="height">
 <rdfs:comment>
   height is a UniqueProperty (can have only one height).
 </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#Player" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="weight">
 <rdfs:comment>
   weight is a UniqueProperty (can have only one weight).
 </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#Player" />
 <rdfs:range rdf:resource
               ="http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="hometown">
 <rdfs:comment>
   hometown is a UniqueProperty (can have only one hometown).
 </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />

 <rdfs:domain rdf:resource="#Player" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>
```

```
<daml:Class rdf:about="#Player">
 <rdfs:subClassOf>
  <daml:Restriction daml:minCardinality="1">
   <daml:onProperty rdf:resource="#name" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>

</rdf:RDF>

            <!-- End of Player Ontology -->
```

DAML Football Ontology

```
            <!-- Football Ontology -->
      <!-- by Kapil Dukle (kapilrd@yahoo.com) -->

<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:daml ="http://www.daml.org/2001/03/daml+oil#"
 xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
 xmlns:event ="http://daml.umbc.edu/ontologies/talk-ont#"
 xmlns:person ="http://daml.umbc.edu/ontologies/person-ont#"
 xmlns:dt ="http://daml.umbc.edu/ontologies/calendar-ont#"
 xmlns:player =
      "http://www.cse.sc.edu/~dukle/ontologies/player-ont.daml#"
 xmlns =
    "http://www.cse.sc.edu/~dukle/ontologies/football-ont.daml#"
>

<daml:Ontology rdf:about="">
 <daml:versionInfo>
   $football-ont.daml, v 1.5, 2001/06/24 02:25:39, Kapil Dukle$
 </daml:versionInfo>
 <rdfs:comment>
   A Football Ontology describing football games, football
   players, player statistics and team statistics.
 </rdfs:comment>
 <daml:imports rdf:resource=
                       "http://www.daml.org/2001/03/daml+oil" />

 <daml:imports rdf:resource=
                   "http://daml.umbc.edu/ontologies/talk-ont" />
 <daml:imports rdf:resource=
                 "http://daml.umbc.edu/ontologies/person-ont" />
 <daml:imports rdf:resource=
    "http://www.cse.sc.edu/~dukle/ontologies/player-ont.daml" />
 <daml:imports rdf:resource=
               "http://daml.umbc.edu/ontologies/calendar-ont" />
 <daml:imports rdf:resource=
                        "http://www.w3.org/2000/01/rdf-schema" />
 <daml:imports rdf:resource=
                  "http://www.w3.org/1999/02/22-rdf-syntax-ns" />
</daml:Ontology>

            <!-- Basic Classes -->

<daml:Class rdf:ID="FootballGame">
 <rdfs:label>Football Game</rdfs:label>
 <rdfs:subClassOf rdf:resource=
             "http://daml.umbc.edu/ontologies/talk-ont#Event" />
</daml:Class>

<daml:Class rdf:ID="PastFootballGame">
 <rdfs:label>Past Game</rdfs:label>
 <rdfs:subClassOf rdf:resource="#FootballGame" />
</daml:Class>

<daml:Class rdf:ID="FutureFootballGame">
 <rdfs:label>Future Game</rdfs:label>
 <rdfs:subClassOf rdf:resource="#FootballGame" />
 <daml:disjointWith rdf:resource="#PastFootballGame" />
</daml:Class>

<daml:Class rdf:ID="Ticket">
 <rdfs:label>Ticket</rdfs:label>
</daml:Class>

<daml:Class rdf:ID="Team">
 <rdfs:label>Team</rdfs:label>
</daml:Class>

<daml:Class rdf:ID="TeamStats">
 <rdfs:label>Team Statistics</rdfs:label>
</daml:Class>

<daml:Class rdf:ID="FootballPlayerStats">
 <rdfs:label>Football Player Statistics</rdfs:label>
</daml:Class>

<daml:Class rdf:ID="OffensivePlayerStats">
 <rdfs:label>Offensive Player Statistics</rdfs:label>
 <rdfs:subClassOf rdf:resource="#FootballPlayerStats" />
</daml:Class>

<daml:Class rdf:ID="QuarterBackStats">
 <rdfs:label>Quarter Back Statistics</rdfs:label>
 <rdfs:subClassOf rdf:resource="#OffensivePlayerStats" />
</daml:Class>

<daml:Class rdf:ID="FullBackStats">
 <rdfs:label>Full Back Statistics</rdfs:label>
 <rdfs:subClassOf rdf:resource="#OffensivePlayerStats" />
</daml:Class>
```

```
<daml:Class rdf:ID="RunningBackStats">
 <rdfs:label>Running Back Statistics</rdfs:label>
 <rdfs:subClassOf rdf:resource="#OffensivePlayerStats" />
</daml:Class>

<daml:Class rdf:ID="TightEndStats">
 <rdfs:label>Tight End Statistics</rdfs:label>
 <rdfs:subClassOf rdf:resource="#OffensivePlayerStats" />
</daml:Class>

<daml:Class rdf:ID="WideReceiverStats">
 <rdfs:label>Wide Receiver Statistics</rdfs:label>
 <rdfs:subClassOf rdf:resource="#OffensivePlayerStats" />
</daml:Class>

<daml:Class rdf:ID="DefensivePlayerStats">
 <rdfs:label>Defensive Player Statistics</rdfs:label>
 <rdfs:subClassOf rdf:resource="#FootballPlayerStats" />
</daml:Class>

<daml:Class rdf:ID="FootballPlayer">
 <rdfs:label>FootballPlayer</rdfs:label>
 <rdfs:subClassOf rdf:resource=
"http://www.cse.sc.edu/~dukle/ontologies/player-ont.daml#Player" />
</daml:Class>

<daml:Class rdf:ID="OffensivePlayer">
 <rdfs:label>OffensivePlayer</rdfs:label>
 <rdfs:subClassOf rdf:resource="#FootballPlayer" />
</daml:Class>

<daml:Class rdf:ID="OffensivePlayerType">
 <rdfs:label>Offensive Player Type</rdfs:label>

 <daml:oneOf rdf:parseType="daml:collection">
  <xsd:string rdf:ID="quarterback" />
  <xsd:string rdf:ID="runningback" />
  <xsd:string rdf:ID="wide_receiver" />
  <xsd:string rdf:ID="tight_end" />
  <xsd:string rdf:ID="fullback" />
  <xsd:string rdf:ID="center" />
  <xsd:string rdf:ID="tackle" />
  <xsd:string rdf:ID="guard" />
 </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="DefensivePlayer">
 <rdfs:label>DefensivePlayer</rdfs:label>
 <rdfs:subClassOf rdf:resource="#FootballPlayer" />
 <daml:disjointWith rdf:resource="#OffensivePlayer" />
</daml:Class>

<daml:Class rdf:ID="DefensivePlayerType">
 <rdfs:label>Defensive Player Type</rdfs:label>
 <daml:oneOf rdf:parseType="daml:collection">
  <xsd:string rdf:value="defensive_tackle" />
  <xsd:string rdf:value="linebacker" />
  <xsd:string rdf:value="defensive_end" />
  <xsd:string rdf:value="defensive_back" />
  <xsd:string rdf:value="nose_tackle" />
 </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="TeamManager">
 <rdfs:label>Team manager</rdfs:label>
 <rdfs:subClassOf rdf:resource=
         "http://daml.umbc.edu/ontologies/person-ont#Person" />
</daml:Class>

<daml:Class rdf:ID="ManagerType">
 <rdfs:label>Manager Type</rdfs:label>
 <daml:oneOf rdf:parseType="daml:collection">
  <xsd:string rdf:ID="teamManager" />
  <xsd:string rdf:ID="teamAsstManager" />
  <xsd:string rdf:ID="teamCoach" />
 </daml:oneOf>
</daml:Class>

            <!-- Properties for FootballGame class -->

<daml:DatatypeProperty rdf:ID="fgTitle">
 <rdfs:comment>
   FootballGame can have one or more titles.
 </rdfs:comment>

 <rdfs:domain rdf:resource="#FootballGame" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fgTournament">
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#FootballGame" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fgStadium">
 <rdfs:comment>
   FootballGame has one stadium. Hence, stadium is a
   UniqueProperty.
 </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#FootballGame" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>
```

```
<daml:DatatypeProperty rdf:ID="fgLeague">
 <rdfs:comment>
   FootballGame has one league. League is a UniqueProperty.
 </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#FootballGame" />
 <rdfs:range rdf:resource=
                 "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fgSponsor">
 <rdfs:domain rdf:resource="#FootballGame" />
 <rdfs:range rdf:resource=
                 "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:ObjectProperty rdf:ID="fgTeam">
 <rdfs:comment> Football game has exactly two teams.
 </rdfs:comment>
 <rdfs:domain rdf:resource="#FootballGame" />
 <rdfs:range rdf:resource="#Team" />
</daml:ObjectProperty>

<daml:Class rdf:about="#FootballGame">
 <rdfs:subClassOf>
  <daml:Restriction daml:minCardinality="1">
   <daml:onProperty rdf:resource="#fgTitle" />
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction daml:cardinality="2">
   <daml:onProperty rdf:resource="#fgTeam" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>

            <!-- Properties for PastFootballGame class -->

<daml:DatatypeProperty rdf:ID="finalScore">
 <rdfs:comment>
   PastGame has exactly one score. Hence, finalScore is a
  UniqueProperty.
 </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#PastFootballGame" />
 <rdfs:range rdf:resource=
                 "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="reviewURL">
 <rdfs:domain rdf:resource="#PastFootballGame" />
 <rdfs:range rdf:resource=
                 "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>
```

```
<daml:DatatypeProperty rdf:ID="attendance">
 <rdfs:domain rdf:resource="#PastFootballGame" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

            <!-- Properties for FutureFootballGame class -->

<daml:DatatypeProperty rdf:ID="telecastStation">
 <rdfs:domain rdf:resource="#FutureFootballGame" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="tvStation">
 <daml:samePropertyAs rdf:resource="#telecastStation"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="gameOdds">
 <rdfs:domain rdf:resource="#FutureFootballGame" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:ObjectProperty rdf:ID="ticket">
 <rdfs:comment> Ticket information. </rdfs:comment>
 <rdfs:domain rdf:resource="#FutureFootballGame" />
 <rdfs:range rdf:resource="#Ticket" />
</daml:ObjectProperty>

            <!-- Properties for Ticket class -->

<daml:DatatypeProperty rdf:ID="ticketURL">
 <rdfs:domain rdf:resource="#Ticket" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="ticketPhone">
 <rdfs:domain rdf:resource="#Ticket" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="ticketPrice">
 <rdfs:domain rdf:resource="#Ticket" />
 <rdfs:range rdf:resource=
               "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

            <!-- Properties for Team class -->

<daml:DatatypeProperty rdf:ID="name">
 <rdfs:comment> name is a UniqueProperty. </rdfs:comment>
 <rdf:type rdf:resource=
       "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#Team" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="hometown">
 <rdfs:comment>
    hometown is a UniqueProperty.
 </rdfs:comment>
 <rdf:type rdf:resource=
       "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#Team" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="mascot">
 <rdfs:comment>
   Team has a single mascot.
 </rdfs:comment>
 <rdf:type rdf:resource=
       "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#Team" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="url">
 <rdfs:domain rdf:resource="#Team" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="website">
 <daml:samePropertyAs rdf:resource="#url"/>
</daml:DatatypeProperty>

<daml:ObjectProperty rdf:ID="teamStats">
 <rdfs:domain rdf:resource="#Team" />
 <rdfs:range rdf:resource="#TeamStats"/>
</daml:ObjectProperty>

            <!-- Properties for TeamStats class -->

<daml:DatatypeProperty rdf:ID="tsSeason">
 <rdfs:comment> The team statistics game season </rdfs:comment>
 <rdfs:domain rdf:resource="#TeamStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>
```

```
<daml:DatatypeProperty rdf:ID="totalFirstDowns">
 <rdfs:comment> The team statistics totalFirstDowns value
 </rdfs:comment>

 <rdfs:domain rdf:resource="#TeamStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="totalOffensiveYards">
 <rdfs:comment> The team statistics totalOffensiveYards value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#TeamStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="totalRushingYards">
 <rdfs:comment> The team statistics totalRushingYards value
</rdfs:comment>
 <rdfs:domain rdf:resource="#TeamStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="totalPassingYards">
 <rdfs:comment> The team statistics totalPassingYards value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#TeamStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="sacks">
 <rdfs:comment> Team stats sacks value </rdfs:comment>
 <rdfs:domain rdf:resource="#TeamStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="touchdowns">
 <rdfs:comment> The team statistics touchdowns value </rdfs:comment>
 <rdfs:domain rdf:resource="#TeamStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:ObjectProperty rdf:ID="timeOfPossession">
 <rdfs:comment> The team statistics timeOfPossession value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#TeamStats" />
 <rdfs:range rdf:resource=
          "http://daml.umbc.edu/ontologies/calendar-ont#time" />

</daml:ObjectProperty>
```

```
            <!-- Properties for FootballPlayerStats class -->

<daml:DatatypeProperty rdf:ID="fpsSeason">
 <rdfs:comment> The football player stats game season
 </rdfs:comment>
 <rdfs:domain rdf:resource="#FootballPlayerStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="g">
 <rdfs:comment> g value </rdfs:comment>
 <rdfs:domain rdf:resource="#FootballPlayerStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="gs">
 <rdfs:comment> gs value </rdfs:comment>
 <rdfs:domain rdf:resource="#FootballPlayerStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

            <!-- Properties for QuarterBackStats class -->

<daml:DatatypeProperty rdf:ID="qbsAttPassing">
 <rdfs:comment> attPassing value </rdfs:comment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="qbsCompPassing">
 <rdfs:comment> compPassing value </rdfs:comment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="qbsPctPassing">
 <rdfs:comment> pctPassing value </rdfs:comment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="qbsYdsPassing">
 <rdfs:comment> ydsPassing value </rdfs:comment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="qbsYpaPassing">
 <rdfs:comment> ypaPassing value </rdfs:comment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
               "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="qbsRatePassing">
 <rdfs:comment> ratePassing value </rdfs:comment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
               "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="qbsAttRushing">
 <rdfs:comment> Quarter back stats attRushing value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="qbsYdsRushing">
 <rdfs:comment> Quarter back stats ydsRushing value
 </rdfs:commment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="qbsAvgRushing">
 <rdfs:comment> Quarter back stats avgRushing value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#QuarterBackStats" />
 <rdfs:range rdf:resource=
               "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

            <!-- Properties for FullBackStats class -->

<daml:DatatypeProperty rdf:ID="fbsAttRushing">
 <rdfs:comment> Full back stats attRushing value </rdfs:comment>
 <rdfs:domain rdf:resource="#FullBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fbsYdsRushing">
 <rdfs:comment>Full back stats ydsRushing value </rdfs:comment>
 <rdfs:domain rdf:resource="#FullBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fbsAvgRushing">
 <rdfs:comment> Full back stats avgRushing value </rdfs:comment>
 <rdfs:domain rdf:resource="#FullBackStats" />
 <rdfs:range rdf:resource=
               "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fbsRecReceiving">
 <rdfs:comment> Full back stats recReceiving value </rdfs:comment>
 <rdfs:domain rdf:resource="#FullBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fbsYdsReceiving">
 <rdfs:comment> Full back stats ydsReceiving value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#FullBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="fbsAvgReceiving">
 <rdfs:comment> Full back stats avgReceiving value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#FullBackStats" />
 <rdfs:range rdf:resource=
               "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

            <!-- Properties for RunningBackStats class -->

<daml:DatatypeProperty rdf:ID="rbsAttRushing">
 <rdfs:comment> Running back stats attRushing value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#RunningBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="rbsYdsRushing">
 <rdfs:comment> Running back stats ydsRushing value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#RunningBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="rbsAvgRushing">
 <rdfs:comment> Running back stats avgRushing value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#RunningBackStats" />
 <rdfs:range rdf:resource=
               "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="rbsRecReceiving">
 <rdfs:comment> Running back stats recReceiving value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#RunningBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="rbsYdsReceiving">
 <rdfs:comment> Running back stats ydsReceiving value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#RunningBackStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="rbsAvgReceiving">
 <rdfs:comment> Running back stats avgReceiving value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#RunningBackStats" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

      <!-- Properties for TightEndStats class -->

<daml:DatatypeProperty rdf:ID="tesRec">
 <rdfs:comment> Tight end stats rec value </rdfs:comment>

 <rdfs:domain rdf:resource="#TightEndStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="tesYds">
 <rdfs:comment> Tight End stats yds value </rdfs:comment>
 <rdfs:domain rdf:resource="#TightEndStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="tesAvg">
 <rdfs:comment> Tight end stats avg value </rdfs:comment>
 <rdfs:domain rdf:resource="#TightEndStats" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<!-- Properties for WideReceiverStats class -->

<daml:DatatypeProperty rdf:ID="wrsRec">
 <rdfs:comment> Wide receiver stats rec value </rdfs:comment>
 <rdfs:domain rdf:resource="#WideReceiverStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="wrsYds">
 <rdfs:comment> Wide receiver stats yds value </rdfs:comment>
 <rdfs:domain rdf:resource="#WideReceiverStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="wrsAvg">
 <rdfs:comment> Wide receiver stats avg value </rdfs:comment>
 <rdfs:domain rdf:resource="#WideReceiverStats" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

            <!-- Properties for DefensivePlayerStats class -->

<daml:DatatypeProperty rdf:ID="dpsSacks">
 <rdfs:comment> Defensive player stats Sacks value
 </rdfs:comment>
 <rdfs:domain rdf:resource="#DefensivePlayerStats" />
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#decimal" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="dpsInt">
 <rdfs:comment> Int value </rdfs:comment>
 <rdfs:domain rdf:resource="#DefensivePlayerStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="dpsYds">
 <rdfs:comment> Defensive Player stats Yds value </rdfs:comment>
 <rdfs:domain rdf:resource="#DefensivePlayerStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="dpsTd">
 <rdfs:comment> td value </rdfs:comment>
 <rdfs:domain rdf:resource="#DefensivePlayerStats" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

            <!-- Properties for FootballPlayer class -->

<daml:DatatypeProperty rdf:ID="number">
 <rdfs:comment> Each FootballPlayer has a single number.
 </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#FootballPlayer" />
 <rdfs:range rdf:resource=
                 "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>
```

```
<daml:DatatypeProperty rdf:ID="fpTeam">
 <rdfs:comment>
   Player can belong to only one team. Hence, team is a
   UniqueProperty.
 </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#FootballPlayer" />
 <rdfs:range rdf:resource=
                 "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="college">
 <rdfs:comment>
   Player can be from more than one college.
 </rdfs:comment>
 <rdfs:domain rdf:resource="#FootballPlayer" />
 <rdfs:range rdf:resource=
                 "http://www.w3.org/2000/10/XMLSchema#string" />
</daml:DatatypeProperty>

<daml:ObjectProperty rdf:ID="draftDate">
 <rdfs:comment> draftDate has at most one value</rdfs:comment>
 <rdfs:domain rdf:resource="#FootballPlayer" />
 <rdfs:range rdf:resource=
          "http://daml.umbc.edu/ontologies/calendar-ont#date" />
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="fpStats">
 <rdfs:comment> draftDate has at most one value</rdfs:comment>
 <rdfs:domain rdf:resource="#FootballPlayer" />
 <rdfs:range rdf:resource="#FootballPlayerStats" />
</daml:ObjectProperty>

<daml:DatatypeProperty rdf:ID="yearsPro">
 <rdfs:comment> yearsPro has at most one value</rdfs:comment>
 <rdfs:domain rdf:resource="#FootballPlayer" />
 <rdfs:range rdf:resource=
     "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger" />
</daml:DatatypeProperty>

<daml:Class rdf:about="#FootballPlayer">
 <rdfs:subClassOf>
  <daml:Restriction daml:minCardinality="1">
   <daml:onProperty rdf:resource="#college" />
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction daml:maxCardinality="1">
   <daml:onProperty rdf:resource="#draftDate" />
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction daml:maxCardinality="1">
   <daml:onProperty rdf:resource="#yearsPro" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>

            <!-- Properties for OffensivePlayer class -->

<daml:ObjectProperty rdf:ID="opStats">
 <rdfs:domain rdf:resource="#OffensivePlayer"/>
 <rdfs:range rdf:resource="#OffensivePlayerStats"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="opType">
 <rdfs:comment> OffensivePlayer needs to of one OffensiveType.
 </rdfs:comment>
 <rdfs:domain rdf:resource="#OffensivePlayer" />
 <rdfs:range rdf:resource="#OffensivePlayerType"/>
</daml:ObjectProperty>

<daml:Class rdf:about="#OffensivePlayer">
 <rdfs:subClassOf>
  <daml:Restriction daml:cardinality="1">
   <daml:onProperty rdf:resource="#opType" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>

            <!-- Properties for DefensivePlayer class -->

<daml:ObjectProperty rdf:ID="dpStats">
 <rdfs:domain rdf:resource="#DefensivePlayer"/>
 <rdfs:range rdf:resource="#DefensivePlayerStats"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="dpType">
 <rdfs:comment> DefensivePlayer needs to of one DefensiveType.
 </rdfs:comment>
 <rdfs:domain rdf:resource="#DefensivePlayer" />
 <rdfs:range rdf:resource="#DefensivePlayerType"/>
</daml:ObjectProperty>

<daml:Class rdf:about="#DefensivePlayer">
 <rdfs:subClassOf>
  <daml:Restriction daml:cardinality="1">
   <daml:onProperty rdf:resource="#dpType" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>

            <!-- Properties for TeamManager class -->

<daml:DatatypeProperty rdf:ID="tenure">
 <rdfs:comment> Can have a single tenure. </rdfs:comment>
 <rdf:type rdf:resource=
        "http://www.daml.org/2001/03/daml+oil#UniqueProperty" />
 <rdfs:domain rdf:resource="#TeamManager"/>
 <rdfs:range rdf:resource=
                "http://www.w3.org/2000/10/XMLSchema#decimal" />
```

```
</daml:DatatypeProperty>

<daml:ObjectProperty rdf:ID="tmType">
 <rdfs:comment> Manager is one of ManagerType </rdfs:comment>
 <rdfs:domain rdf:resource="#TeamManager" />
 <rdfs:range rdf:resource="#ManagerType"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="tmTeam">
 <rdfs:domain rdf:resource="#TeamManager" />
 <rdfs:range rdf:resource="#Team"/>
</daml:ObjectProperty>

<daml:Class rdf:about="#TeamManager">
 <rdfs:subClassOf>
  <daml:Restriction daml:cardinality="1">
   <daml:onProperty rdf:resource="#type" />
  </daml:Restriction>
 </rdfs:subClassOf>
<rdfs:subClassOf>
  <daml:Restriction daml:cardinality="1">
   <daml:onProperty rdf:resource="#team" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>

</rdf:RDF>

            <!-- End of Football Ontology -->
```

Jess Football Rules

```
;; Jess rules created by Kapil Dukle (kapilrd@yahoo.com)


;; Rule 1: A FootballPlayer can play for only one team

(defrule one-team
 "A FootballPlayer can play for a single team"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:FootballPlayer)
 ?t1 <- (PropertyValue football:fpTeam  ?n        ?a)
 ?t2 <- (PropertyValue football:fpTeam  ?n        ?b)
 ?t3 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#type
             ?a
             http://www.w3.org/2000/10/XMLSchema#string)
 ?t4 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#type
             ?b
             http://www.w3.org/2000/10/XMLSchema#string)
 ?t5 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?a
             ?name1)
 ?t6 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?b
             ?name2)
 (test (> (call ?t2 getFactId) (call ?t1 getFactId)))
 (test (> (call ?t4 getFactId) (call ?t3 getFactId)))
 (test (> (call ?t6 getFactId) (call ?t5 getFactId)))
 =>
 (printout WSTDOUT "Violated Rule:A FootballPlayer can play for only
                    one team" crlf)
 (printout WSTDOUT ?n " plays for " ?name2 crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t2 getFactId)  crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t4 getFactId) crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t6 getFactId) crlf)
 (retract ?t2)
 (retract ?t4)
 (retract ?t6) )


;; Test Assertion for Rule 1

(assert(PropertyValue
            football:fpTeam
            players-in:Adams_Keith
            players-in:_a1))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            players-in:_a1
            http://www.w3.org/2000/10/XMLSchema#string))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            players-in:_a1
            "Chicago_Bears"))



;; Rule 2: A DefensivePlayer can have only one player type

(defrule one-def-type
 "A DefensivePlayer can have only one player type"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:DefensivePlayer)
 ?t1 <- (PropertyValue football:dpType  ?n        ?a)
 ?t2 <- (PropertyValue football:dpType  ?n        ?b)
 ?t3 <- (PropertyValue
             http://www.w3.org/2000/10/XMLSchema#string
             ?a
             ?a1)
 ?t4 <- (PropertyValue
             http://www.w3.org/2000/10/XMLSchema#string
             ?b
             ?b1)
 ?t5 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?a1
             ?type1)
 ?t6 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?b1
             ?type2)
 (test (> (call ?t2 getFactId) (call ?t1 getFactId)))
 (test (> (call ?t4 getFactId) (call ?t3 getFactId)))
 (test (> (call ?t6 getFactId) (call ?t5 getFactId)))
 =>
 (printout WSTDOUT "Violated Rule:A DefensivePlayer can have only one
                    player type" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t2 getFactId) crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t4 getFactId) crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t6 getFactId) crlf)
 (retract ?t2)
 (retract ?t4)
 (retract ?t6) )


;; Test Assertion for Rule 2

(assert(PropertyValue
            football:dpType
            players-in:Allen_Will
            players-in:_t1))
(assert(PropertyValue
            http://www.w3.org/2000/10/XMLSchema#string
            players-in:_t1
            players-in:_t2))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            players-in:_t2
            "center"))



;; Rule 3: An OffensivePlayer can have only one player type

(defrule one-off-type
 "An OffensivePlayer can have only one player type"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:OffensivePlayer)
 ?t1 <- (PropertyValue football:opType  ?n        ?a)
 ?t2 <- (PropertyValue football:opType  ?n        ?b)
 ?t3 <- (PropertyValue
             http://www.w3.org/2000/10/XMLSchema#string
             ?a
             ?a1)
 ?t4 <- (PropertyValue
             http://www.w3.org/2000/10/XMLSchema#string
             ?b
             ?b1)
 ?t5 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?a1
             ?type1)
 ?t6 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?b1
             ?type2)
 (test (> (call ?t2 getFactId) (call ?t1 getFactId)))
 (test (> (call ?t4 getFactId) (call ?t3 getFactId)))
 (test (> (call ?t6 getFactId) (call ?t5 getFactId)))
 =>
 (printout WSTDOUT "Violated Rule:An OffensivePlayer can have only one
                   player type" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t2 getFactId)  crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t4 getFactId) crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t6 getFactId) crlf)
 (retract ?t2)
 (retract ?t4)
 (retract ?t6) )


;; Test Assertion for Rule 3

(assert(PropertyValue
            football:opType
            players-in:Alexander_Shaun
            players-in:_t1))
(assert(PropertyValue
            http://www.w3.org/2000/10/XMLSchema#string
            players-in:_t1
            players-in:_t2))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            players-in:_t2
            "nose_tackle"))



;; Rule 4: A FootballGame consists of 2 different teams

(defrule two-diff-team-rule
 "A FootballGame consists of 2 different teams"
 ?t1 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#type
             ?n
             football:FootballGame)
 ?t2 <- (PropertyValue
             football:fgTeam
             ?n
             ?team1)
 ?t3 <- (PropertyValue
             football:name
             ?team1
             ?n1)
 ?t4 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#type
             ?n1
             http://www.w3.org/2000/10/XMLSchema#string)
 ?t5 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?n1
             ?name1)
 ?t6 <- (PropertyValue
             football:fgTeam
             ?n
             ?team2)
 ?t7 <- (PropertyValue
             football:name
             ?team2
             ?n2)
 ?t8 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#type
             ?n2
             http://www.w3.org/2000/10/XMLSchema#string)
 ?t9 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?n2
             ?name2)
 (test (<> (call ?t5 getFactId) (call ?t9 getFactId) ))
 (test (= 0 (str-compare ?name1 ?name2)))
 =>
 (printout WSTDOUT "Violated Rule:A FootballGame consists of 2
                    different teams" crlf)
 (printout WSTDOUT "Team 1:" ?name1 " Team 2:" ?name2 crlf)
 (printout WSTDOUT "Retracting all facts about the game:" ?n "[" (call
                    ?t1 getFactId) ", " (call ?t2 getFactId) ", "
                   (call ?t3 getFactId) ", " (call ?t4 getFactId) ", "
                   (call ?t5 getFactId) ", " (call ?t6 getFactId) ","
                   (call ?t7 getFactId) ", " (call ?t8 getFactId) ", "
                   (call ?t9 getFactId) "]" crlf)
 (retract ?t1)
 (retract ?t2)
 (retract ?t3)
 (retract ?t4)
 (retract ?t5)
 (retract ?t6)
 (retract ?t7)
 (retract ?t8)
 (retract ?t9) )
```

```
;; Test Assertion for Rule 4

(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            games-inst:Pittsburgh_Steelers-Pittsburgh_Steelers
            football:FootballGame))
(assert(PropertyValue
            football:fgTeam
            games-inst:Pittsburgh_Steelers-Pittsburgh_Steelers
            games-inst:Pittsburgh_Steelers_8))
(assert(PropertyValue
            football:fgTeam
            games-inst:Pittsburgh_Steelers-Pittsburgh_Steelers
            games-inst:Pittsburgh_Steelers_9))
(assert(PropertyValue
            football:name
            games-inst:Pittsburgh_Steelers_8
            games-inst:arp1))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            games-inst:arp1
            http://www.w3.org/2000/10/XMLSchema#string))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            games-inst:arp1
            "Pittsburgh_Steelers"))
(assert(PropertyValue
            football:name
            games-inst:Pittsburgh_Steelers_9
            games-inst:arp2))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            games-inst:arp2
            http://www.w3.org/2000/10/XMLSchema#string))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            games-inst:arp2
            "Pittsburgh_Steelers"))



;; Rule 5: A FootballGame consists of exactly 2 teams

(defrule two-team-rule
 "A FootballGame consists of exactly 2 teams"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:FootballGame)
 ?t1 <- (PropertyValue football:fgTeam ?n ?team1)
 ?t2 <- (PropertyValue football:fgTeam ?n ?team2)
 ?t3 <- (PropertyValue football:fgTeam ?n ?team3)
 ?t4 <- (PropertyValue football:name ?team3 ?n3)
 ?t5 <- (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#value
              ?n3
              ?val)
 (test (<> (call ?t1 getFactId) (call ?t2 getFactId)))
 (test (<> (call ?t2 getFactId) (call ?t3 getFactId)))
 (test (> (call ?t3 getFactId) (call ?t1 getFactId)))
 (test (> (call ?t3 getFactId) (call ?t2 getFactId)))
 =>
 (printout WSTDOUT "Violated Rule:A FootballGame consists of exactly 2
                    teams" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t3 getFactId) crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t4 getFactId) crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t5 getFactId) crlf)
 (retract ?t3)
 (retract ?t4)
 (retract ?t5) )


;; Test Assertion for Rule 5

(assert(PropertyValue
            football:fgTeam
            games-inst:Indianapolis_Colts-New_York_Jets-2
            games-inst:Pittsburgh_Steelers_2))
(assert(PropertyValue
            football:name
            games-inst:Pittsburgh_Steelers_2
            games-inst:arp2))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            games-inst:arp2
            "Pittsburgh_Steelers"))



;; Rule 6: A Football Player's age is always more than his years as a
professional

(defrule age-rule
 "A Football Player's age is always more than his years as a
professional"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:FootballPlayer)
 (PropertyValue player:age  ?n  ?a)
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#value
      ?a
      ?age)
 (PropertyValue football:yearsPro   ?n    ?b)
 ?t1 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?b
             ?yrspro)
 (test (> ?yrspro ?age))
 =>
 (printout WSTDOUT "Violated Rule:A Football Player’s age is always
                    more than his years as a professional" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t1 getFactId) crlf)
 (printout WSTDOUT "Asserting yearsPro value to 0" crlf)
 (retract ?t1)
 (assert(PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?b
             0))
)


;; Test Assertion for Rule 6

(assert (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#type
             players-in:Test_Player
             football:FootballPlayer))
(assert(PropertyValue
            player:age
            players-in:Test_Player
            players-in:_a1))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            players-in:_a1
            29))
(assert(PropertyValue
            football:yearsPro
            players-in:Test_Player
            players-in:_a2))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            players-in:_a2
            30))



;; Rule 7: Two teams cannot have the same mascot

(defrule one-mascot-rule
 "Two teams cannot have the same mascot"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n1
      football:Team)
 (PropertyValue football:mascot     ?n1   ?m1)
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?m1
      http://www.w3.org/2000/10/XMLSchema#string)
 ?t1 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?m1
             ?val1)
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n2
      football:Team)
 (PropertyValue football:mascot      ?n2    ?m2)
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?m2
      http://www.w3.org/2000/10/XMLSchema#string)
 ?t2 <- (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#value
              ?m2
              ?val2)
 (test (<> (call ?t1 getFactId) (call ?t2 getFactId)))
 (test (= 0 (str-compare ?val1 ?val2)))
 =>
 (printout WSTDOUT "Violated Rule:Two teams cannot have the same
                   mascot" crlf)
 (printout WSTDOUT ?n1 "=" ?val1 crlf)
 (printout WSTDOUT ?n2 "=" ?val2 crlf)
 ;; (Insert the action here - Retract new team mascot, or set value to
 ;;    null)
)


;; Test Assertion for Rule 7

(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            teams-inst:Test_Team
            football:Team))
(assert(PropertyValue
            football:mascot
            teams-inst:Test_Team
            teams-inst:arp1))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            teams-inst:arp1
            http://www.w3.org/2000/10/XMLSchema#string))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            teams-inst:arp1
            "Vikings"))



;; Rule 8: Two teams cannot have the same website

(defrule one-website-rule
 "Two teams cannot have the same website"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n1
      football:Team)
 (PropertyValue football:website     ?n1    ?m1)
 ?t1 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?m1
             ?val1)
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n2
      football:Team)
 (PropertyValue football:website     ?n2    ?m2)
 ?t2 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?m2
             ?val2)
 (test (<> (call ?t1 getFactId) (call ?t2 getFactId)))
 (test (= 0 (str-compare ?val1 ?val2)))
 =>
 (printout WSTDOUT "Violated Rule:Two teams cannot have the same
                    website" crlf)
 (printout WSTDOUT ?n1 "=" ?val1 crlf)
 (printout WSTDOUT ?n2 "=" ?val2 crlf)
 ;; (Insert the action here - Retract new team website, or set value to
 ;; null)
)


;; Test Assertion for Rule 8

(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            teams-inst:Test_Team
            football:Team))
(assert(PropertyValue
            football:website
            teams-inst:Test_Team
            teams-inst:arp1))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            teams-inst:arp1
            http://www.w3.org/2000/10/XMLSchema#string))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            teams-inst:arp1
            "http://www.vikings.com/"))



;; Rule 9: A Defensive Player cannot have statistics values less than 0

(defrule positive-defstats-rule
 "A Defensive Player cannot have statistics values less than 0"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:DefensivePlayer)
 (PropertyValue football:dpStats     ?n     ?s)
 (PropertyValue    ?p                   ?s    ?arp)
 ?t1 <- (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#value
              ?arp
              ?val)
 (test (> 0 ?val))
 =>
 (printout WSTDOUT "Violated Rule:A Defensive Player cannot have
                    statistics values less than 0" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t1 getFactId) " for "  ?p
                    "=" ?val crlf)
 (printout WSTDOUT "Asserting "  ?p "= 0"  crlf)
 (retract ?t1)
 (assert(PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#value
              ?arp
              0)) )


;; Test Assertion for Rule 9
(assert(PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#type
              players-in:Test_Player
              football:DefensivePlayer))
(assert(PropertyValue
              football:dpStats
              players-in:Test_Player
              players-in:Test_Player_2003))
(assert(PropertyValue
              football:dpsTd
              players-in:Test_Player_2003
              players-in:arp1))
(assert(PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#value
              players-in:arp1
              -1))



;; Rule 10: An Offensive Player cannot have statistics values less than
0

(defrule positive-offstats-rule
 "An OffensivePlayer cannot have statistics values less than 0"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:OffensivePlayer)
 (PropertyValue football:opStats     ?n    ?s)
 (PropertyValue    ?p                ?s    ?arp)
 ?t1 <- (PropertyValue
              http://www.w3.org/1999/02/22-rdf-syntax-ns#value
              ?arp
              ?val)
 (test (> 0 ?val))
 =>
 (printout WSTDOUT "Violated Rule:An Offensive Player cannot have
                    statistics values less than 0" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t1 getFactId) " for "  ?p
                    "=" ?val crlf)
 (printout WSTDOUT "Asserting "  ?p "= 0"  crlf)
 (retract ?t1)
 (assert(PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?arp
             0)) )


;; Test Assertion for Rule 10

(assert(PropertyValue
          http://www.w3.org/1999/02/22-rdf-syntax-ns#type
          players-in:Test_Player
          football:OffensivePlayer))
(assert(PropertyValue
            football:opStats
            players-in:Test_Player
            players-in:Test_Player_2003))
(assert(PropertyValue
            football:gs
            players-in:Test_Player_2003
            players-in:arp1))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            players-in:arp1
            -1))



;; Rule 11 : A Past Football Game cannot have a begin year later than
the current year

(defrule past-game-year-rule
 "A Past Football Game cannot have a begin year later than the current
year"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:PastFootballGame)
 (PropertyValue  event:BeginTime  ?n  ?bt)
 (PropertyValue  dt:Year  ?bt  ?arp)
 ?t1 <- (PropertyValue
             http://www.w3.org/1999/02/22-rdf-syntax-ns#value
             ?arp
             ?val)
 (test (> ?val 2003))
 =>
 (printout WSTDOUT "Violated Rule:A Past Football Game cannot have a
                    begin year later than the current year" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t1 getFactId) " for
                    dt:Year=" ?val crlf)
 (printout WSTDOUT "Asserting  dt:Year = 2003"  crlf)
 (retract ?t1)
 (assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            ?arp
            2003)) )


;; Test assertion for Rule 11

(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            games-inst:Test_Game
            football:PastFootballGame))
(assert(PropertyValue
            event:BeginTime
            games-inst:Test_Game
            games-inst:arp1))
(assert(PropertyValue
            dt:Year
            games-inst:arp1
            games-inst:arp2))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            games-inst:arp2
            2005))



;; Rule 12 : A Future Football Game cannot have a begin year earlier
than the current year

(defrule future-game-year-rule
 "A Future Football Game cannot have a begin year earlier than the
current year"
 (PropertyValue
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      ?n
      football:FutureFootballGame)
 (PropertyValue  event:BeginTime  ?n  ?bt)
 (PropertyValue  dt:Year  ?bt  ?arp)
 ?t1 <-  (PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            ?arp
            ?val)
 (test (< ?val 2003))
 =>
 (printout WSTDOUT "Violated Rule:A Future Football Game cannot have a
                    begin year earlier than the current year" crlf)
 (printout WSTDOUT "Retracting fact:" (call ?t1 getFactId) " for
                    dt:Year=" ?val crlf)
 (printout WSTDOUT "Asserting dt:Year = 2003"  crlf)
 (retract ?t1)
 (assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            ?arp
            2003)) )
```

```
;; Test assertion for Rule 12

(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            games-inst:Test_Game
            football:FutureFootballGame))
(assert(PropertyValue
            event:BeginTime
            games-inst:Test_Game
            games-inst:arp1))
(assert(PropertyValue
            dt:Year
            games-inst:arp1
            games-inst:arp2))
(assert(PropertyValue
            http://www.w3.org/1999/02/22-rdf-syntax-ns#value
            games-inst:arp2
            2001))
```

MyDAMLParser.java

```
import java.util.*;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.jena.daml.common.DAMLModelImpl;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.jena.daml.common.DAMLClassImpl;
import com.hp.hpl.mesa.rdf.jena.common.*;
import com.hp.hpl.mesa.rdf.jena.mem.ModelMem;

import edu.drexel.gicl.daml.*;


/**
 * MyDAMLParser.java
 * <p>
 * Contains methods to parse a DAML/RDF
 * ontology. This class utilizes the Jena
 * toolkit (v1.6.1). Used by the parser to
 * display classes, properties and other
 * information in an ontology.
 *
 *
 * @author   Kapil Dukle (kapilrd@yahoo.com)
 *
 */
```

```
public class MyDAMLParser {

      /** RDF Literal Constant
       */
      private static final String LITERAL = "Literal";

      /** RDFS Domain Constant
       */
      private static final String RDFS_DOMAIN =
                         "http://www.w3.org/2000/01/rdf-schema#domain";

      /** Default Ontology
       */
      private static final String DEFAULT_ONT =
          "http://www.cse.sc.edu/~dukle/ontologies/football-ont.daml#";

      /** Default base URI
       */
      private static final String DEFAULT_URI =
                            "http://www.cse.sc.edu/~dukle/ontologies/";

      /** Default ontologies to be loaded
       */
      public static final String[] defLoadFiles =
                              {"player-ont.daml", "football-ont.daml"};

      /** Current uri
       */
      private String uri;

      /** Hashtable containing fully qualified names of classes
       *  and their corresponding properties
       */
      private Hashtable classTable;

      /** Vector containing local names of DAML classes
       */
      private Vector damlClasses;



      /** Vector containing all object properties defined
       *  by the ontology
       */
      private Vector objProperties;

      /** DAML files loaded into the inference engine (including
       *  all DAML instance files).
       */
      private Vector loadedFiles;

      /** Model object to store DAML/RDF ontology information
       */
      private DAMLModel model;
      /** MyDAMLParser constructor
       *
       */
      public MyDAMLParser() {
            this(DEFAULT_ONT);
      }

      /** MyDAMLParser constructor
       *
       *
       * @param  uri - the uri to be parsed
       *
       */
      public MyDAMLParser(String uri) {
            this.uri = uri;
      }

      /*
       * Reads the ontology at the uri, and stores information
       * about DAML classes, properties, and object properties.
       *
       *
       * @param
       *
       * @return true - success
       *         false - parsing failed
       *
       */
      public boolean parseClassInfo() {
        if(loadedFiles != null)
          loadedFiles.removeAllElements();

        loadedFiles = new Vector(getDefaultLoadedFiles());

        // Clear all previous vectors

        if(classTable != null)
          classTable.clear();

        if(damlClasses != null)
          damlClasses.removeAllElements();

        if(objProperties != null)
          objProperties.removeAllElements();

        classTable = new Hashtable();
        damlClasses = new Vector();
        objProperties = new Vector();

        model = new DAMLModelImpl();

        // Read the ontology specified by the uri
        try {
          model.read(uri);
        }
        catch(Exception e) {
```

```
        System.out.println("Exception reading model");
        return false;
      }
      catch(Error e) {
        System.out.println("Error reading model");
        return false;
      }

      Iterator i = model.listDAMLClasses();

      while(i.hasNext()) {
          try {
            DAMLClass dc = (DAMLClass)i.next();
            String name = dc.getLocalName();
            String dcns = dc.getNameSpace();

            // Ignore classes from standard namespaces

            if( (dcns.equals(NameSpaces.DAML_NS))
                ||(dcns.equals(NameSpaces.RDF_NS))
                ||(dcns.equals(NameSpaces.RDFS_NS)) )
              continue;

            Vector props = getPropsForClass(dc);

            // Store DAML classes and their properties

            classTable.put(new String(NameSpaces.getNSPrefix(dcns)
               + NameSpaces.NS_SEPARATOR + name), new Vector(props));

            damlClasses.add(dc);
          }
          catch(NullPointerException e) {
            //e.printStackTrace(System.err);
            continue;
          }
        }

        return true;
    }


    /*
     * Returns a list of properties for a DAML class
     *
     *
     * @param  dc - the DAMLClasses
     *
     * @return Vector containing list of properties
     *                 for dc
     *
     */
    private Vector getPropsForClass(DAMLClass dc) {
      Vector props = new Vector();

      Iterator i = dc.getDefinedProperties(true);
      while(i.hasNext()) {
        try {
          DAMLProperty dp = (DAMLProperty)i.next();

          String ns = dp.getNameSpace();
          String name = dp.getLocalName();

          // Ignore classes from standard namespaces

          if( (ns.equals(NameSpaces.DAML_NS))
                ||(ns.equals(NameSpaces.RDF_NS))
                ||(ns.equals(NameSpaces.RDFS_NS)) )
            continue;

          String pref = NameSpaces.getNSPrefix(ns);

          RDFNode node = dp.prop_range().get();

          String propName = ((ResourceImpl)node).getLocalName();

          // Check if property is an Object Property

          if((!(dp instanceof DAMLDatatypeProperty))
                            && (!(propName.equals(LITERAL)))) {
            objProperties.add( pref + NameSpaces.NS_SEPARATOR +
                                                            name);
          }

          props.add(new String(pref + NameSpaces.NS_SEPARATOR
                                                         + name));

        } catch(NullPointerException e) {
            e.printStackTrace(System.err);
            continue;
          }
      }

      props.trimToSize();

      return props;
    }


    /*
     * Returns a list of properties for a predicate
     * that is a DAML Object Property or that has
     * another Resource as the RDFS range value
     *
     *
     * @param  predicate - the Object Property
     *
     * @return Vector containing list of properties
     *                for predicate class (for object properties
     *                only)
     *
```

```
 */
public Vector getPropsOfPredicate(String predicate) {
  Vector result = new Vector();

  String uri = NameSpaces.getExpandedURI(predicate);

  String ns = uri.substring(0, uri.lastIndexOf("#"));

  Iterator i = model.listDAMLProperties();

  while(i.hasNext()) {
    try {
      DAMLProperty dp = (DAMLProperty)i.next();

      // Test for our required property

      if(!(dp.getLocalName().equals
                     (uri.substring(uri.lastIndexOf("#") + 1))))
        continue;

      PropertyAccessor pa = dp.prop_range();

      RDFNode node = pa.get();

      String resName = ((ResourceImpl)node).getLocalName();

      // Make sure property is an object property

      if((!(dp instanceof DAMLDatatypeProperty))
                        && (!(resName.equals(LITERAL)))) {

        // Get the range resources for this property

        Iterator j = dp.getRangeClasses();

        if(j.hasNext()) {
          Object o = j.next();

          // Range is a DAML Class

          if(o instanceof DAMLClassImpl) {

          // Get defined properties for the DAML Class

      Iterator k =((DAMLClassImpl)o).getDefinedProperties(true);

          while(k.hasNext()) {
            DAMLProperty p = (DAMLProperty)k.next();

            String propns = p.getNameSpace();

            // Ignore properties from standard namespaces

            if( (propns.equals(NameSpaces.DAML_NS))
                ||(propns.equals(NameSpaces.RDF_NS))
                ||(propns.equals(NameSpaces.RDFS_NS)) )
                continue;

              result.add(NameSpaces.getNSPrefix(propns) +
                         NameSpaces.NS_SEPARATOR
                         + p.getLocalName());

          }
        }

        // Range is an RDF Resource

        else if(o instanceof ResourceImpl) {

          // Get properties for ResourceImpl

        result = getResourceProperties((ResourceImpl)o, resName);

        }
      }

      break;
    }
  } catch(Exception e) {
      e.printStackTrace(System.err);
      //System.out.println("Returning null");
      return null;
    }
  }

  result.trimToSize();

  return result;
}


/*
 * Sets the current uri (to be parsed)
 *
 *
 * @param  uri - the uri to be parsed
 *
 * @return void
 *
 */
public void setURI(String uri) {
  if((uri == null) || (uri.trim().length() == 0)) {
    this.uri = DEFAULT_ONT;
    return;
  }

  this.uri = uri;
}
```

```
    /*
     * Returns the current uri
     *
     *
     * @param
     *
     * @return String representing the current uri
     *
     */
    public String getURI() {
      return uri;
    }


    /*
     * Returns the default ontology uri
     *
     *
     * @param
     *
     * @return String representing the
     *                  default ontology
     *
     */
    public String getDefaultOntology() {
      return DEFAULT_ONT;
    }


    /*
     * Returns a list of default DAML files loaded in
     * the knowledge base (for the current
     * ontology)
     *
     *
     * @param
     *
     * @return Vector containing list of
     *                  default loaded files
     *
     */
    private Vector getDefaultLoadedFiles() {
      Vector v = new Vector();

      if(!uri.endsWith("#"))
        uri += "#";

      // Load the files specified by defLoadFiles array
      // for a default ontology

      if(uri.equals(DEFAULT_ONT)) {
        for(int i=0; i<defLoadFiles.length; i++) {
          String str = defLoadFiles[i];
          if(!str.endsWith("#"))
            str += "#";
            v.addElement(DEFAULT_URI + str);
          }
        }
        else {
          v.addElement(uri);
        }

        v.trimToSize();

        return v;
    }


    /*
     * Returns a list of all DAML files loaded in
     * the knowledge base (for the current
     * ontology)
     *
     *
     * @param
     *
     * @return Vector containing list of all
     *                 loaded files
     *
     */
    public Vector getLoadedFiles() {
      return loadedFiles;
    }


    /*
     * Returns a list of all object properties
     * in the current ontology
     *
     *
     * @param
     *
     * @return Vector containing list of object
     *                 properties
     *
     */
    public Vector getObjectProperties() {
      return objProperties;
    }


    /*
     * Deletes the list of files specified by the
     * parameter from the list of currently loaded
     * files
     *
     *
     * @param  errfiles - Vector containing list of
     *              files to be removed
     *
     * @return void
     *
```

```
 */
public void delFromLoadedFiles(Vector errfiles) {
  int index;

  for(int i=0; i<errfiles.size(); i++) {
    String str = (String)errfiles.elementAt(i);
    if((index = loadedFiles.indexOf(str)) != -1)
      loadedFiles.removeElementAt(index);
    }

    loadedFiles.trimToSize();
}


/*
 * Adds the file specified by the parameter to
 * the list of loaded files
 *
 *
 * @param  file - file to be added to the list
 *
 * @return void
 *
 */
public void updateLoadedFiles(String file) {
  if(loadedFiles.indexOf(file) != -1)
    return;

  loadedFiles.add(file);
}


/*
 * Returns a list of all DAML classes
 *
 *
 * @param
 *
 * @return Vector containing all the DAML
 *                classes
 *
 */
public Vector getAllClassesVector() {
  Vector result = new Vector();

  try {
    if(classTable == null)
      parseClassInfo();

      // Get all classes from the hashtable

      for (Enumeration e = classTable.keys() ;
                                 e.hasMoreElements() ;) {
        result.add(e.nextElement());
      }
```

```
        result.trimToSize();
    } catch(Exception e) { }

    return result;
  }


  /*
   * Returns a list of predicates for
   * the class specified by the parameter
   *
   *
   * @param  name - the DAML class name
   *
   * @return Vector containing list of
   *              predicates for the DAML class
   *              specified by 'name'
   *
   */
  public Vector getPredicateForClass(String name) {
    Vector pv = null;
    try {
      if(classTable == null)
        parseClassInfo();

      // Get the predicate for class stored in
      // the hashtable

      pv = (Vector)classTable.get(name);
      pv.trimToSize();

    } catch(Exception e) { }

    return pv;
  }


  /*
   * Returns a list containing the subclasses and
   * the superclasses for the DAML class specified
   * by the parameter
   *
   *
   * @param  name - the DAML class name
   *
   * @return Vector containing list of
   *              predicates for the DAML class
   *              specified by 'name'
   *
   */
  public Vector getClassHierarchyVector(String classname) {
    if((classname == null) || ((classname =
                              classname.trim()).length() == 0))
      return null;

    int index = classname.indexOf(NameSpaces.NS_SEPARATOR);
```

```
if(index != -1)
  classname = classname.substring(index+1);

Vector result = null;

if(damlClasses == null)
  parseClassInfo();

int size = damlClasses.size();
DAMLClass dc = null;

for(int i=0; i<size; i++) {
  try {
    dc = (DAMLClass)damlClasses.elementAt(i);

    // Find the DAML class

    if(dc.getLocalName().equalsIgnoreCase(classname)) {
      String ns = dc.getNameSpace();
      result = new Vector();
      result.add(NameSpaces.getNSPrefix(ns) +
                  NameSpaces.NS_SEPARATOR + dc.getLocalName());
      break;
    }

  } catch(NullPointerException e) { continue; }
}

// Get its subclasses and superclasses

Iterator subIter = dc.getSubClasses(true);
Iterator superIter = dc.getSuperClasses(true);


while(subIter.hasNext()) {
  try {
    Object o = subIter.next();
    if(!(o instanceof DAMLClass))
      continue;

    DAMLClass subClass = (DAMLClass)o;
    String ns = subClass.getNameSpace();

    // Ignore standard namespace resources

    if( (ns.equals(NameSpaces.DAML_NS))
          ||(ns.equals(NameSpaces.RDF_NS))
          ||(ns.equals(NameSpaces.RDFS_NS)) )
      continue;

    result.add(NameSpaces.getNSPrefix(ns) +
            NameSpaces.NS_SEPARATOR + subClass.getLocalName());

  } catch(NullPointerException e) { continue; }
}
```

```
    while(superIter.hasNext()) {
      try {
        Object o = superIter.next();
        if(!(o instanceof DAMLClass))
          continue;

        DAMLClass superClass = (DAMLClass)o;
        String ns = superClass.getNameSpace();

        // Ignore standard namespace resources

        if( (ns.equals(NameSpaces.DAML_NS))
             ||(ns.equals(NameSpaces.RDF_NS))
             ||(ns.equals(NameSpaces.RDFS_NS)) )
          continue;

        result.add(NameSpaces.getNSPrefix(ns) +
             NameSpaces.NS_SEPARATOR + superClass.getLocalName());

      } catch(NullPointerException e) { continue; }
    }

    result.trimToSize();

    return result;
  }



  /*
   * Returns a list of properties for the resource given
   * by resName
   *
   *
   * @param  ri - Class representing the Resource
   * @param  resName - String representing the Resource name
   *
   * @return Vector containing list of properties for Resource ri
   *
   */
  private Vector getResourceProperties(ResourceImpl ri,
                                               String resName
    Vector result = new Vector();

    String resNS = ri.getNameSpace();
    String nuri = resNS;

    Model m = new ModelMem();

    // Read the resource name space

    try {
      m.read(resNS);
    } catch(RDFException e) {
        e.printStackTrace(System.err);
```

```
          if((e.getNestedException()) instanceof
                                  java.io.FileNotFoundException) {

            // If file not found, retry after
            // adding the ".daml" suffix

            if(resNS.endsWith("#"))
              nuri = resNS.substring(0, resNS.lastIndexOf('#'));

            nuri += ".daml";
            nuri += "#";

            try {
              m.read(nuri);
            }  catch(Exception ex) { return result; }

            ri = new ResourceImpl(nuri+resName, m);
          }
        }

      try {
        Property p = new PropertyImpl(RDFS_DOMAIN);

        // Get all statements that have a range value
        // (p) of resource ri

        StmtIterator si = m.listStatements(new
                       SelectorImpl((Resource)null, p, (RDFNode)ri));

        while(si.hasNext()) {
          Statement st = (Statement)si.next();

          Resource subject = st.getSubject();


          result.add(NameSpaces.getNSPrefix(subject.getNameSpace())
                                   + NameSpaces.NS_SEPARATOR
                                   + subject.getLocalName());
        }
      }  catch(Exception e) {
           e.printStackTrace(System.err);
         }


      return result;
    }


    /*
     * Testing the class
     *
     *
     * @param args - an array containing command line arguments
     *
     * @return void
     *
```

```
	 */
	public static void main(String[] args) {

	}
}
```

MyDAMLJessKB.java

```


import java.util.Vector;
import java.util.Iterator;
import java.util.ArrayList;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.jena.daml.common.DAMLModelImpl;
import com.hp.hpl.mesa.rdf.jena.model.*;

import jess.*;

import edu.drexel.gicl.daml.damljesskb.*;
import edu.drexel.gicl.daml.*;


/**
 * MyDAMLJessKB.java
 * <p>
 * Uses DAMLJessKB to run the inference engine
 * (Jess). It also contains methods to generate Jess
 * queries in response to user inputs, and retrieve
 * query results.
 *
 *
 * @author   Kapil Dukle (kapilrd@yahoo.com)
 *
 */

public class MyDAMLJessKB {

	/** Newline constant
	 */
	public static final String NEWLINE =
	                            System.getProperty("line.separator");

	/** Jess query name
	 */
	private static final String queryName = "search";

	/** Jess result variable
	 */
	private static final String resultVar = "RESULT";
```

```
/** DAMLJessKB object
 */
private DAMLJessKB damljesskb;

/*
 * Loads DAML files as facts into the inference
 * engine
 *
 *
 * @param  damljesskb - object that loads facts into
 *                         the inference engine
 * @param  filenames - DAML files to be loaded
 *
 * @return Vector containing files that caused errors
 *             while loading
 *
 */
public static Vector loadDAMLFiles(DAMLJessKB damljesskb,
                                              Vector fileNames) {
  Vector loadErrors = null;
  int size = fileNames.size();

  if(size == 0)
    return null;

  String uri = "";

  for(int i=0; i<size; i++) {
    try {
      System.out.println("---> Loading:" + (uri =
                             (String)fileNames.elementAt(i)));

      damljesskb.loadDAMLResource(uri);
    } catch(Error e) {
        System.err.println("---> Error loading:" + uri);
        System.err.println("Error class:" +
                                         e.getClass().getName());
        e.printStackTrace(System.err);

        if(loadErrors == null)
          loadErrors = new Vector();
          loadErrors.add(uri);
          continue;
      }
      catch(Exception e) {
        System.out.println("---> Exception loading:" + uri);
        System.err.println("Exception class:" +
                                         e.getClass().getName());
        e.printStackTrace(System.err);
        if(loadErrors == null)
          loadErrors = new Vector();
        loadErrors.add(uri);
        continue;
      }
  }
```

```
    return loadErrors;
  }


  /*
   * Prints all facts in the knowledge base
   *
   *
   * @param  damljesskb - object that is an interface to
   *                              the Jess knowledge base
   * @param  countOnly - If true, prints fact count only
   *                            If false, prints facts as well
   *                            as fact count
   *
   * @return int value representing the number of facts
   *
   */
  public static int printAllFacts(DAMLJessKB damljesskb,
                                              boolean countOnly) {
    System.out.println();
    System.out.println("--- List Facts");

    short count = 0;

    Iterator iter = damljesskb.listFacts();

    while(iter.hasNext()) {
      Object o = iter.next();

      if(o instanceof jess.Fact) {
        if(!countOnly) {
          System.out.println("" + count + ": " + o.toString());

          System.out.println();
        }
        count++;

      }
    }

    System.out.println("Total of " + count + " facts!");

    return count;
  }


  /*
   * Runs the Jess query
   *
   *
   * @param  query - the Jess query
   * @param  rete - the inference engine
   *
   * @return String representing the query result
   *
   */
```

```
      public static String runQuery(String query, Rete rete) {
        String result = "No matches found";
        ArrayList nlist = null;

        try {
          System.out.println("Query:" + query);

          // Execute the Jess query

          rete.executeCommand(query);

          // Store results in a Jess result variable

          rete.store(resultVar, rete.runQuery(queryName, new
                                                     ValueVector()));

          rete.executeCommand("(store " + resultVar + "(run-query " +
                                                   queryName + "))");

          nlist = MyDAMLJessKB.getQueryResults(rete, resultVar);

          result = MyDAMLJessKB.getQueryResultsStr(nlist);

        } catch(Exception e) {
            e.printStackTrace(System.err);
          }
          catch(Error e) {
            e.printStackTrace(System.err);
          }

        return result;
      }


      /*
       * Generates Jess query using subject, predicate, predicate
       * properties (for object properties), operator, and search value
       *
       *
       * @param  subject - the DAML class
       * @param  predicate - the DAML property of the class
       * @param  propOfPred - property of predicate (for object
properties
       *                                 only)
       * @param  operator - comparison operator
       * @param  value - search value
       * @param  isObjectProperty - true if predicate is an object
property,
       *                                          false otherwise
       *
       * @return String representing the Jess query
       *
       */
      public static String generateQuery(String subject,
                            String predicate, String propOfPred,
                             String operator, String value,
                                      boolean isObjectProperty) {
        String test = "";

        // If operator is null, ignore value

        if(operator == null) {
          operator = "";
          value = "";
        }
        else {
          boolean isNumeric = true;

          // Test if the value is a number

          try {
            double dVal = Double.parseDouble(value);

          } catch(NumberFormatException e) {
              isNumeric = false;
            }

          if(isNumeric)
            test = "&:(or (integerp ?s) (floatp ?s))" +  "&:(" +
                               operator + " (float ?s) " + value + ")";
          else
            test = "&:(" + operator + " (str-compare ?s " + "\"" +
                                                 value + "\"" + ") 0)";

        }

        StringBuffer queryBuf = new StringBuffer();

        queryBuf.append("(defquery " + queryName + NEWLINE);
        queryBuf.append("(declare (max-background-rules 100))" +
                                                              NEWLINE);
        queryBuf.append("(PropertyValue " + NameSpaces.RDF_NS + "type
                                      ?n " + subject + ") " + NEWLINE);


        queryBuf.append("(PropertyValue " + predicate + " ?n ?res) " +
                                                              NEWLINE);

      // If predicate is an object property, include the propOfPred in
      // the query

      if(isObjectProperty) {
        if(propOfPred == null)
          propOfPred = "?x";

        queryBuf.append("(PropertyValue " + propOfPred + " ?res ?y) " +
                                                              NEWLINE);
        queryBuf.append("(PropertyValue " + NameSpaces.RDF_NS + "value
                                                  ?y ?s" + test + ")");
      }
      else
        queryBuf.append("(PropertyValue " + NameSpaces.RDF_NS + "value
                                                ?res ?s" + test + ")");

        queryBuf.append(" )" + NEWLINE);

        return queryBuf.toString();
      }



      /*
       * Returns query results as an ArrayList
       *
       *
       * @param  rete - the inference engine
       * @param  result - the Jess result variable
       *
       * @return ArrayList containing the results
       *
       */
      public static ArrayList getQueryResults(Rete rete,
                                                       String result) {
        ArrayList v = null;

        try {
          // Fetch the result variable

          Iterator i =
               (Iterator)rete.fetch(result).externalAddressValue(null);

          v = new ArrayList();

          ValueVector multislot = null;
          Fact f = null;
          StringBuffer buf = null;
          Token t = null;

          while(i.hasNext()) {

            // Each value in the result is represented by jess.Token

            t = (Token)i.next();

            // First fact that contains the matched resource

            buf = new StringBuffer();

            f = t.fact(1);
            multislot = f.get(0).listValue(null);

            buf.append(multislot.get(1).stringValue(null));

            buf.append(NEWLINE + "\t");

            // Penultimate fact contains the matched property

            f = t.fact(t.size() - 2);
            multislot = f.get(0).listValue(null);

            buf.append(multislot.get(0).stringValue(null));

            buf.append(" = ");

            // Last fact contains the actual value

            f = t.fact(t.size() - 1);
            multislot = f.get(0).listValue(null);

            buf.append(multislot.get(2).stringValue(null));

            v.add(buf.toString());

            buf = null;
          }

        } catch(Error e) {
            e.printStackTrace(System.err);
          }
          catch(Exception e) {
            e.printStackTrace(System.err);
          }

        return v;
      }


      /*
       * Returns query results as a String
       *
       *
       * @param  v - ArrayList containing the results
       *
       * @return String representing the query results
       *
       */
      public static String getQueryResultsStr(ArrayList v) {
        if((v==null) || (v.size()==0)) {
          return "0 matches found";
        }

        // Printing answers

        int count = 0;
        Iterator iter = v.iterator();

        StringBuffer sb = new StringBuffer();

        while(iter.hasNext())  {
          count++;
          sb.append("" + count + "\t" + iter.next() + NEWLINE);
        }

        return ("Search returned " + count + " matches" + NEWLINE +
                                                       sb.toString());
      }


      /*
       * Testing the class
       *
       *
       * @param args - an array containing command line arguments
       *
       * @return void
       *
       */

      public static void main(String[] args) {

      }

}
```