ISSN:0975-9646

Architecting and Designing of Semantic Web Based Application using the JENA and PROTÉGÉ – A Comprehensive Study

Archana P. Kumar, Kumar Abhishek, and Vipin Kumar.N
Department of Computer Science and Engineering, MIT Manipal-576104

Abstract: **The Resource Description Format (RDF) is used to represent information modeled as a "graph": a set of individual objects, along with a set of connections among those objects. In that role, RDF is one of the pillars of the so-called Semantic Web. This paper describes how the RDF graph can be modeled by using the Java based framework called Jena. Jena based on Java deals with programmatic statements. The same can be done by using an editor- Protégé. The paper compares how the semantic web concepts can be designed and modeled using the two API and also states as to which API should be used while developing the Semantic Based Web Applications for better performance metrics .**
Keywords: **Jena, Protégé, OWL, RDF, Semantic.**

I. INTRODUCTION

The Semantic Web is a "web of data" that enables machines to understand the semantics, or meaning, of information on the World Wide Web.[1] It extends the network of hyperlinked human-readable web pages by inserting machine-readable metadata about pages and how they are related to each other, enabling automated agents to access the Web more intelligently and perform tasks on behalf of users. The term was coined by Tim Berners-Lee,[2] the inventor of the World Wide Web and director of the World Wide Web Consortium, which oversees the development of proposed Semantic Web standards. He defines the Semantic Web as "a web of data that can be processed directly and indirectly by machines."

The term "Semantic Web" is often used more specifically to refer to the formats and technologies that enable it.[3] These technologies include the Resource Description Framework (RDF), a variety of data interchange formats (e.g. RDF/XML, N3, Turtle, N-Triples), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain.

A) JENA FRAMEWORK

Jena is a Java framework for building Semantic Web applications[1]. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and grown out of work with the HP Labs Semantic Web Programme.

The Jena Framework includes:

- A RDF API
- Reading and writing RDF in RDF/XML, N3 and N-Triples
- An OWL API
- In-memory and persistent storage
- RDQL- a query language for RDF
- SPARQL query engine

Jena is a programming toolkit, using the Java Programming Language. While there are a few command-line tools to help perform some key tasks using Jena, mostly Jena is used by writing Java Programs[2].

Jena is a Java API which can be used to create and manipulate RDF graphs. Jena has object classes to represent graphs, resources, properties and literals. The interfaces representing resources, properties and literals are called Resource, Property and Literal respectively.

In Jena, a graph is called a model and is represented by the Model Interface.

To build applications exploiting the ontology, we need an API allowing us to access and manipulate directly an ontology written in OWL. Only a few exist and nearly all of them are based on the Jena framework. Jena is a Java framework for building semantic web applications. It is open source and provides various programming toolboxes- an OWL API. Since it is reliable, mature and offers a good compatibility with most of the other RDFS/OWL API, Jena was the choice of API to build the required application.

Thus Jena Classes can be used to[4]:

- Retrieve and Parse an RDF File containing a graph or a collection of graphs.
- Store it in memory
- Examine each triple in turn, examine one component(say, the subject) of each triple in turn, or examine only triples that meet specified criteria and
- Write a serialized version of a graph to a file.

An RDF Graph is stored in Jena as a “model”, and a Jena model is created by a factory, as in:

Model m=ModelFactory.createDefaultModel();

Once a model has been defined, Jena can populate it by reading data from files, backend databases, etc. in various formats and once it has been populated, Jena can perform set operations on pairs of populated models and /or search models for specific values or combinations (patterns) of values.

B) RESOURCE DESCRIPTION FRAMEWORK (RDF)

The Resource Description Framework (RDF) is a standard (technically a W3C Recommendation) for describing resources[5]. A Resource is anything we can identify for example a class Pizza, Ingredient, Menu,etc. Every Resource has a URI, a Universal Resource Identifier. A URI can be a URL or some other kind of unique identifier.

RDF is best thought of in the form of node and arc diagrams. Each arc in an RDF Model is called a statement. Each

statement asserts a fact about a resource. A statement has three parts:

- the *subject* is the resource from which the arc leaves
- the *predicate* is the property that labels the arc
- the *object* is the resource or literal pointed to by the arc

A statement is sometimes called a triple, because of its three parts.

A simple vcard might look like this in RDF:

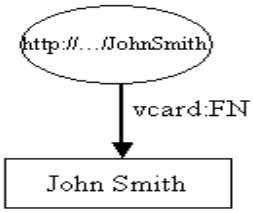

Fig. 2 : simple vcard in rdf with more details of the person node

The *resource*, John Smith, is shown as an elipse and is identified by a Uniform Resource Identifier (URI)[1], in this case "http://.../JohnSmith". Resources have *properties*. In these examples the sort of properties that would appear on John Smith's business card. Figure 1 shows only one property, John Smith's full name. Properties are usually represented in this qname form when written as RDF XML and it is a convenient shorthand for representing them in diagrams and in text. Strictly, however, properties are identified by a URI.

Each property has a value. In this case the value is a *literal*, which for now we can think of as a strings of characters. Literals are shown in rectangles.

Jena is a Java API which can be used to create and manipulate RDF graphs like this one. Jena has object classes to represent graphs, resources, properties and literals. The interfaces representing resources, properties and literals are called Resource, Property and Literal respectively. In Jena, a graph is called a model and is represented by the Model interface.

The code to create this graph, or model, is simple:

```
// some definitions
static String personURI    = "http://somewhere/JohnSmith";
static String fullName     = "John Smith";

// create an empty Model
Model model = ModelFactory.createDefaultModel();

// create the resource
Resource johnSmith = model.createResource(personURI);

// add the property
johnSmith.addProperty(VCARD.FN, fullName);
```

The John Smith resource is then created and a property added to it. The property is provided by a "constant" class VCARD which holds objects representing all the definitions in the VCARD schema. Jena provides constant classes for other well known schemas, such as RDF and RDF schema themselves, Dublin Core and DAML.

The code to create the resource and add the property, can be more compactly written in a cascading style:

```
Resource johnSmith =
model.createResource(personURI)
.addProperty(VCARD.FN, fullName);
```

Another example of representing different parts of John Smith's name can be:

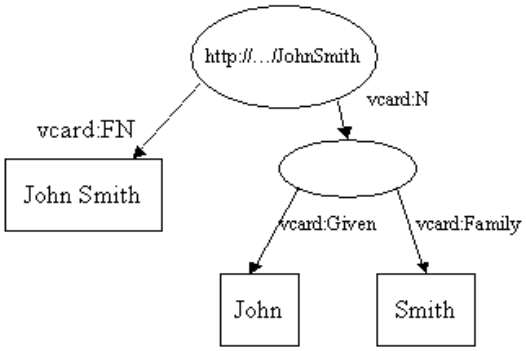

Fig. 2 : simple vcard in rdf with more details of the person node

The Jena code to construct this example, is again very simple. First some declarations and the creation of the empty model.

```
// some definitions
String personURI    = "http://somewhere/JohnSmith";
String givenName    = "John";
String familyName   = "Smith";
String fullName     = givenName + " " + familyName;

// create an empty Model
Model model = ModelFactory.createDefaultModel();

// create the resource
//   and add the properties cascading style
Resource johnSmith
= model.createResource(personURI)
.addProperty(VCARD.FN, fullName)
.addProperty(VCARD.N,
model.createResource()
.addProperty(VCARD.Given, givenName)
.addProperty(VCARD.Family, familyName));
```

Jena has methods for reading and writing RDF as XML. These can be used to save an RDF model to a file and later read it back in again.

The key RDF package for the application developer is com.hp.hpl.jena.rdf.model. The API has been defined in terms of interfaces so that application code can work with different implementations without change. This package contains interfaces for representing models, resources, properties, literals, statements and all the other key concepts of RDF, and a ModelFactory for creating models. So that application code remains independent of the implementation, it is best if it uses interfaces wherever possible, not specific class implementations.

II. PROTEGE

Protégé is a free open source Ontology Editor and Knowledge Based Framework developed and published by Stanford University[6]. It is published under the terms of Mozilla Public License[7].

Protégé is a flexible, configurable platform for the development of arbitrary model-driven applications and components. Protégé has an open architecture that allows programmers to integrate plug-ins, which can appear as separate tabs, specific user interface components (widgets), or perform any other task on the current model. The Protégé-OWL editor provides many editing and browsing facilities for OWL models, and therefore can serve as an attractive starting point for rapid application development. Developers can initially wrap their components into a Protégé tab widget and later extract them to distribute them as part of a stand-alone application.

The Protégé-OWL editor enables users to:

- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties, and SWRL (Semantic Web Rule Language) rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoners such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup.

The Protégé API not only has a non-visual model part, but also comes with comprehensive support for user interface programming. There are several convenient classes and utility methods that help programmers develop interactive user interfaces quickly and with uniform look-and-feels that match the rest of the Protégé family of tools.

One of the foundations of UI programming with Protégé is the event mechanism, which allows programmers to react cleanly on changes.

- Protégé is a free, open-source platform to construct domain models and knowledge based applications with ontologies.
- Ontologies range from taxonomies, classifications, database schemas to fully axiomatized theories.
- Ontologies are now central to many applications such as scientific knowledge portals, information management and integration systems, electronic commerce and web services.

There are two main ways of modelling ontologies:

- Frame-based
- OWL

Each of them have its own user interface.

- Protégé Frames Editor: enables users to build and populate ontologies that are frame-based, in accordance with OKBC (Open Knowledge Base Connectivity Protocol).
  - Classes
  - Slots for properties and relationships
  - Instances for class
- Protégé OWL Editor: enables users to build ontology for the Semantic Web, in particular to OWL
  - Classes
  - Properties
  - Instances
  - *Reasoning*

III. WEB ONTOLOGY LANGUAGES (OWL)

Ontologies are used to capture knowledge about some domain of interest. Ontology describes the concepts in the domain and also the relationships that hold between those concepts. Different ontology languages provide different facilities. The most recent development in standard ontology languages is OWL from the World Wide Web Consortium (W3C) [8]. Like Protégé, OWL makes it possible to describe concepts but it also provides new facilities. It has a richer set of operators - e.g. intersection, union and negation. It is based on a different logical model which makes it possible for concepts to be defined as well as described[9,10].

Complex concepts can therefore be built up in definitions out of simpler concepts. Furthermore, the logical model allows the use of a reasoner which can check whether or not all of the statements and definitions in the ontology are mutually consistent and can also recognise which concepts fit under which definitions. The reasoner can therefore help to maintain the hierarchy correctly. This is particularly useful when dealing with cases where classes can have more than one parent. OWL ontologies have similar components to Protégé frame based ontologies. However, the terminology used to describe these components is slightly different from that used in Protégé . OWL ontology consists of Individuals, Properties, and Classes, which roughly correspond to Protégé Instances, Slots and Classes.

Individuals are the objects. An important difference between Protégé and OWL is that OWL does not use the Unique Name Assumption (UNA). This means that two different names could actually refer to the same individual. For example, “Queen Elizabeth”, “The Queen” and “Elizabeth Windsor” might all refer to the same individual. In OWL, it must be explicitly stated that individuals are the same as each other, or different to each other — otherwise they might be the same as each other, or they might be different to each other. Individuals are also known as instances. Individuals can be referred to as being ‘instances of classes’. Properties are binary relations on individuals - i.e. properties link two individuals together. For example, the property hasSibling might link the individual Matthew to the individual Gemma, or the property hasChild might link the individual Peter to the individual Matthew. Properties can have inverses. For example, the inverse of hasOwner is isOwnedBy. Properties can be limited to having a single value i.e. to being functional. They can also be either transitive or symmetric. Properties are roughly equivalent to slots in Protégé.

OWL has two main types of properties: Object properties and Datatype properties.

- Object properties relate an individual to an individual.
- Datatype properties link an individual to a data value.
- A third type of property, Annotation properties, can be used to attach ‘meta-data’ to classes, properties and individuals.

OWL supports the specification of a property hierarchy.

- We can specify that a property has a super-property. In fact, for any given property we can specify multiple super properties.

- In OWL, object properties may only have object properties as super-properties, and datatype properties may only have datatype properties as super-properties.

OWL classes are interpreted as sets that contain individuals. They are described using formal (mathematical) descriptions that state precisely the requirements for membership of the class. For example, the class Cat would contain all the individuals that are cats in the domain of interest. Classes may be organised into a superclass-subclass hierarchy, which is also known as taxonomy. Subclasses specialise ('are subsumed by') their superclasses. OWL supports six main ways of describing classes of individuals. The simplest of these is a Named Class. The other types of class descriptions are anonymous classes.

- Named Classes – create a class and assign a name to it. Two 'built in' named classes: owl:Thing and owl:Nothing.
- Anonymous classes- built up from class descriptions.
- Intersection, Union and Complement classes.
- Restriction classes- existential, universal, cardinality, hasValue.
- Enumeration classes.
- Combinations of Named classes and anonymous classes are used to build up complex class descriptions.

IV. Difference Between Jena and Protégé

Indeed, a shortcoming of the Jena Framework for OWL exploitation comes from its very nature: Jena is a general RDF/RDFS framework. Thus Jena lacks specific primitives for OWL-based applications. It is the opposite with the Protégé-OWL API which is dedicated to OWL manipulation and provides most functions needed to exploit OWL ontology. This results in a faster and simpler programming. Moreover, since this API is powering the Protégé ontology editor, it benefits from the same development support as the editor and is not likely to be forsaken any time soon. So after considering the pros and cons of the different API, the Protégé-OWL API is the final choice for the programming needs. It is worth noting that these API being Java-based, this implies at least the core of the applications to be coded in Java.

V. Conclusion

The Protégé-OWL API is centered on a model of ontologies. More precisely, the API includes classes that describe every OWL item (concept, property, etc.) and the model is an instantiation of the whole ontology. In this model each OWL item is represented as an instance of its corresponding class. This ontology model described by the class *OWLModel* is not the only one existing. Not only each OWL API has its own model, but more importantly each reasoner has one too. In our implementation we use the OWLModel from the Protégé-OWL API and the reasoner uses a translation of it in its own optimized format.

Jena uses different encoding <?xml version='1.0' encoding='iso-8859-1'?> and more recent RDF-S specification than in Protégé <ENTITY rdfs 'http:www.w3.org/2000/01/rdf-schema#'>[11.]

Both are APIs and can be used for similar tasks with the only main difference that Protégé-OWL is based on a much older framed-based API which predates OWL and RDF. Therefore Protégé had to do many design compromise which was found inconvenient. Jena on the other hand has been designed for RDF and OWL from the start and is optimized for handling triples, queries, etc.

The Protégé-OWL used Jena for parsing and provides a Jena "view" (implementation of the Graph interface) so that some Jena services can be exploited for Protége.

It can be easy to create an ontology file with Protégé and read that into a Jena Model and then process it as required.

The Choice of which APRI to used should be determined by two factors as given below:

- Does the API offer a set of programming abstractions that can be used comfortably to implement the user requriements.
- Is the API actively supported and bug fixed in case the standards changes.

TABLE1 DIFFERENCE BETWEEN PROTÉGÉ AND JENA.

Protégé	**Jena**
User Friendly in terms of user interface	Does not have a user interface.
Has forms which can be used to insert records.	Has a programming code to enter the properties and their values.
Does not involve any programming code.	Involves only programming code.
Has reasoner and inbuilt query tabs to form query to the database.	Can work with SPARQL and RDQL which are query languages to query the RDF Data.
Uses a much older version of specification and encoding than Jena.	Uses very recent encoding and specification as compared to Protégé.
Has the specifications based on OWL.	Lacks the primitives based on OWL.

References

[1] "W3C Semantic Web Frequently Asked Questions". W3C. http://www.w3.org/2001/sw/SW-FAQ. Retrieved March 13, 2008.

[2] Berners-Lee, Tim; James Hendler and Ora Lassila (May 17, 2001). "The Semantic Web". Scientific American Magazine. http://www.sciam.com/article.cfm?id=the-semantic-web&print=true. Retrieved March 26, 2008.

[3] Herman, Ivan (March 12, 2008). "W3C Semantic Web Activity". W3C. http://www.w3.org/2001/sw/. Retrieved March 13, 2008.

[4] JENA http://jena.sourceforge.net/tutorial/RDF_API/index.html.

[5] http://jena.sourceforge.net/tutorial/RDF_API/

[6] Chaoqing Lv, Takashi Kobayashi, Kiyoshi Agusa, Kun Wu, Qing Zhu

[7] Matthew Horridge,Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe: A

[8] Practical Guide To Building OWL Ontologies Using The Protégé-OWL

[9] Tutorial:http://protege.stanford.edu/conference/2005/slides/T2_OWLTutorialI