JENA –A SEMANTIC WEB TOOL

by
Ranjani Sankaran
&
krishna Priyanka Chebrolu

RDF

- Resource Descriptor Framework
- W3C Standard for describing resources on the web.
- Is designed to provide common way to describe information so it can be read and understood by computer application.

- What is Jena
- Capabilities of Jena
- RDF Implementation using Jena

What is Jena

- Jena is a Java framework for the creation of applications for the Semantic Web.
- Provides interfaces and classes for the creation and manipulation of RDF repositories.
- Also provides classes/interfaces for the management of OWL-based ontologies

Capabilities of Jena

- RDF API
- Reading and writing in RDF/XML, N-Triples
- OWL API
- In-memory and persistent storage
- SPARQL query engine

RDF Implementation using Jena

- Resources, Properties, Literals, Statements (Triples: <subj predicate obj>)
- A set of (related) statements constitute an RDF graph.
- The Jena RDF API contains classes and interfaces for every important aspect of the RDF specification.
- They can be used in order to construct RDF graphs from scratch, or edit existent graphs.
- These classes/interfaces reside in the com.hp.hpl.jena.rdf.model package.
- In Jena, the Model interface is used to represent RDF graphs. Through the Model, statements can be obtained/ created/ removed etc.

RDF API -Example

- Hello World Example:

```
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.datatypes.xsd.XSDDatatype;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.Resource;
public class HelloRDFWorld{
    public static void main(String[] args){
        Model m =ModelFactory.createDefaultModel();
        String NS="http://example.com/test/";
        Resource r=m.createResource(NS+"r");
        Property p=m.createProperty(NS+ "p");
        r.addProperty(p,"hello world",XSDDatatype.XSDstring);
        m.write(System.out,"Turtle");

    }
}
```

RDF API – Example cond..

- // Namespace declarations

static final String *familyUri = "http://tinman.cs.gsu.edu#";*

- // Create an empty Model
  model = ModelFactory.*createDefaultModel();*
  model.setNsPrefix("student", "http://tinman.cs.gsu.edu#");
- // Create an empty Model
  model = ModelFactory.*createDefaultModel();*
  model.setNsPrefix("student", "http://tinman.cs.gsu.edu#");
- // Create resources representing the people in our model
  Resource ranjani = model.createResource(*familyUri +"Ranjani");*

.

RDF API-Example Cont..

- // Add properties to describing the relationships between them
  ranjani.addProperty(fullname,"Ranjani Sankaran");
- // Statements can also be directly created ...
  Statement statement1 =
  model.createStatement(ranjani,college,"gsu");
- // ... then added to the model:
  model.add(statement1);
- // Arrays of Statements can also be added to a Model:
  Statement statements[] = **new Statement[5];**
  statements[0] = model.createStatement(ranjani,major,"csc");

model.add(statements);

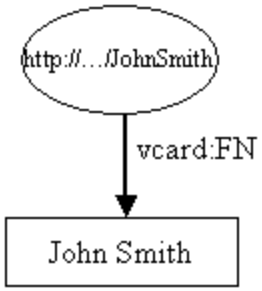
RDF API-Contd..

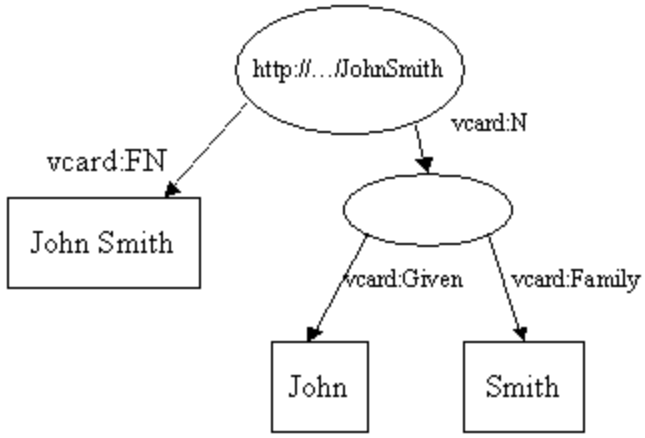
- // A List of Statements can also be addedList list = new ArrayList();
  list.add(model.createStatement(ranjani,course ,"DB and the Web"));
- model.add(list);

  model.write(new PrintWriter(System.*out));*

RDF Representation of Vcard

- An alternative RDF mapping for the format defined by vCard





Querying RDF using JENA

```
//List students who have taken a course
   ResIterator students_course = model.listSubjectsWithProperty(course);

   // Because subjects of statements are Resources, the method returned a
     ResIterator
   while (students_course.hasNext()) {

     // ResIterator has a typed nextResource() method
     Resource person = students_course.nextResource();

     // Print the URI of the resource
     System.out.println("The list of students who have taken
     coures"+person.getURI());
   }
```

Querying RDF using JENA contd..

```
// To find all the courses taken by a student, the model itself
   can be queried
   NodeIterator moreStudents1 =
    model.listObjectsOfProperty(priyanka, course);
   System.out.println("****LIST OF COURSES TAKEN BY
    PRIYANKA***");
   while (moreStudents1.hasNext()) {

   System.out.println(moreStudents1.nextNode().toString());
    }
```

RDF Validator

- Validates the format of RDF documents created using JENA

```
public static void main(String args[])
    {
          Model data = FileManager.get().loadModel("student.rdf");
       InfModel infmodel = ModelFactory.createRDFSModel(data);
       ValidityReport validity = infmodel.validate();
       if (validity.isValid()) {
          System.out.println("OK");
       } else {
          System.out.println("Conflicts");
          for (Iterator i = validity.getReports(); i.hasNext(); ) {
             System.out.println(" - " + i.next());
```

JENA ONTOLOGY MODEL

- An ontology model is an extension of the Jena RDF model that provides extra capabilities for handling ontologies. Ontology models are created through the Jena ModelFactory. The simplest way to create an ontology model is as follows:
- OntModel m = ModelFactory.createOntologyModel(); This will create an ontology model with the *default* settings, which are set for maximum compatibility with the previous version of Jena.

OWL API

```
// Create an empty ontology model
OntModel ontModel = ModelFactory.createOntologyModel();
String ns = new String("http://www.example.com/onto1#");
String baseURI = new String("http://www.example.com/onto1");
Ontology onto = ontModel.createOntology(baseURI);


// Create 'Person', 'MalePerson' and 'FemalePerson' classes
OntClass person = ontModel.createClass(ns + "Person");
OntClass malePerson = ontModel.createClass(ns + "MalePerson");
OntClass femalePerson = ontModel.createClass(ns + "FemalePerson");


// FemalePerson and MalePerson are subclasses of Person
person.addSubClass(malePerson);
person.addSubClass(femalePerson);


// FemalePerson and MalePerson are disjoint
malePerson.addDisjointWith(femalePerson);
femalePerson.addDisjointWith(malePerson);
```

OWL API-Properties

```
// Create datatype property 'hasAge'
DatatypeProperty hasAge =
ontModel.createDatatypeProperty(ns + "hasAge");
// 'hasAge' takes integer values, so its range is 'integer'
// Basic datatypes are defined in the ‘vocabulary’ package
hasAge.setDomain(person);
hasAge.setRange(XSD.integer); // com.hp.hpl.jena.vocabulary.XSD


// Create individuals
Individual john = malePerson.createIndividual(ns + "John");
Individual jane = femalePerson.createIndividual(ns + "Jane");
Individual bob = malePerson.createIndividual(ns + "Bob");


// Create statement 'John hasAge 20'
Literal age20 =
ontModel.createTypedLiteral("20", XSDDatatype.XSDint);
Statement johnIs20 =
ontModel.createStatement(john, hasAge, age20);
ontModel.add(johnIs20);
```

Reasoning

- Jena is designed so that inference engines can be ‘plugged’ in Models and reason with them.
- The reasoning subsystem of Jena is found in the com.hp.hpl.jena.reasoner package.
- All reasoners must provide implementations of the ‘Reasoner’ Java interface
- Once a Reasoner object is obtained, it must be ‘attached’ to a Model. This is accomplished by modifying the Model specifications

Reasoning-OWL example

- A sample schema and a data file are taken.
- We can create an instance of the OWL reasoner, specialized to the schema and then apply that to the data to obtain an inference model.

Reasoning-Example

```
Model schema =
   FileManager.get().loadModel("owlDemoSchema.
   owl");
   Model data =
   FileManager.get().loadModel("owlDemoData.rdf"
   );
   Reasoner reasoner =
   ReasonerRegistry.getOWLReasoner();
   reasoner = reasoner.bindSchema(schema);
   InfModel infmodel =
   ModelFactory.createInfModel(reasoner, data);
```

SPARQL query processing

- Jena uses the ARQ engine for the processing of SPARQL queriesThe ARQ API classes are found in com.hp.hpl.jena.query
- Basic classes in ARQ:Query: Represents a single SPARQL query.
- Dataset: The knowledge base on which queries are executed (Equivalent to RDF Models)
- QueryFactory: Can be used to generate Query objects from SPARQL strings
- QueryExecution: Provides methods for the execution of queries
- ResultSet: Contains the results obtained from an executed query
- QuerySolution: Represents a row of query results.
  - If there are many answers to a query, a ResultSet is returned after the query is executed. The ResultSet contains many QuerySolutions

SPARQL QUERY-Example

```
// Create a new query
  String queryString =
   "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> "+
     "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  "+
     "select ?uri "+
     "where { "+
      "?uri rdfs:subClassOf <http://www.opentox.org/api/1.1#Feature>
  "+
     "} \n ";
  com.hp.hpl.jena.query.Query query =
   QueryFactory.create(queryString);
```

References

- Introduction to JENA

- Jena Ontology API

http://jena.sourceforge.net/ontology/#creatingModels