Programming the Semantic Web

Tutorials

Jena

The OWL API

protégé

Outline

- Part 1: Apache Jena:
  - query online RDF stores (Triplestores)
  - combine reasoning rules
- Part 2: Protégé OWL/OWL API
  - read/write OWL ontology
  - edit classes and properties
  - create/edit instances

Ontology and Java code used in this tutorial

- Ontology
  - University.owl
- Java code files
  - OWLAPI.java (Protégé-OWL)
  - SPARQL_Dbpedia.java (Apache Jena)
- The complete source code can be found here:
  - Blackboard
  - http://www.cs.le.ac.uk/people/yh37/SW_API.zip

Part 1: Querying RDF stores using Apache Jena

- Download Apache Jena from
  - https://jena.apache.org/download/index.cgi
- Document Overview
  - https://jena.apache.org/documentation/
- Javadoc
  - https://jena.apache.org/documentation/javadoc/jena/

Part 1: SPARQL Endpoints

- Querying RDF stores (Triplestores)
  - A service endpoint is a (referenceable) entity, processor, or resource to which Web service messages can be addressed. [W3C]
  - A SPARQL endpoint enables application to query a knowledge base (RDF stores/Triplestores) via SPARQL language
  - Allow applications to send queries using SPARQL or to use HTTP operations (RESTful)
  - For example, DBpedia has a SPARQL endpoint at http://dbpedia.org/sparql (You can also visit it using your browser)
  - A list a SPARQL endpoints
    - https://www.w3.org/wiki/SparqlEndpoints

Part 1: Querying RDF stores using Apache Jena (1)

```
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.rdf.model.RDFNode;
…
String queryString = ”SELECT ?s ?p ?o WHERE {?s ?p ?o} LIMIT 100";

    Query query = QueryFactory.create(queryString);
    QueryExecution qexec =
    QueryExecutionFactory.sparqlService("http://dbpedia.org/sparql", query);
    ResultSet results = qexec.execSelect();

      while(results.hasNext()){
            QuerySolution qs = results.nextSolution();
            ……
    }
```

Dbpedia SPARQL endpoint

Part 1: Querying RDF stores using Apache Jena (2)

- Print the headers

```
.........

while(results.hasNext()){

        QuerySolution qs = results.nextSolution();
        List<String> columnNames=results.getResultVars();

        //print the headers

        for(String heading: columnNames){
                System.out.print(heading+"\t");
        }
      .........
}
 ....
```

Part 1: Querying RDF stores using Apache Jena (3)

- Iterate over the rows

```
.....
for(int column=0;column<columnNames.size();column++){
  RDFNode node=qs.get(columnNames.get(column));

        if(node.isAnon()){ //if this RDFNode is an anonymous resource.
                System.out.print("anonymous:");
        }else if(node.isURIResource()){//if this RDFNode is an URI resource
                System.out.print("URI:");
        }else if(node.isLiteral()){//if this RDFNode is a Literal
                System.out.print("literal:");
        }else if(node.isResource()){//if this node is a resource
                System.out.print("resource:");
        }else{                          //else
        }
        System.out.print(node.toString()+"\t");
}
....
```

Part 1: More Examples

- List all classes in the triplestore.

```
String queryString = "SELECT ?class ?label ?description\n”+
                "WHERE {\n”+
                "  ?class a owl:Class.\n ”+
                "  optional { ?class rdfs:label ?label}\n ”+
                "  optional { ?class rdfs:comment ?description}\n ”+
                "} "
```

- Search for Leicester’s twin towns on **Dbpedia**.

```
String queryString = "PREFIX dep:<http://dbpedia.org/resource/>\n"+
                "PREFIX dbo:<http://dbpedia.org/ontology/>\n"+
                                "SELECT ?town ?country WHERE {\n"+
                "dep:Leicester dbo:twinTown ?town."+
                "?town dbo:country ?country."+
"} LIMIT 10";
```

Part 1: Combine SPARQL with Jena Reasoning Rules

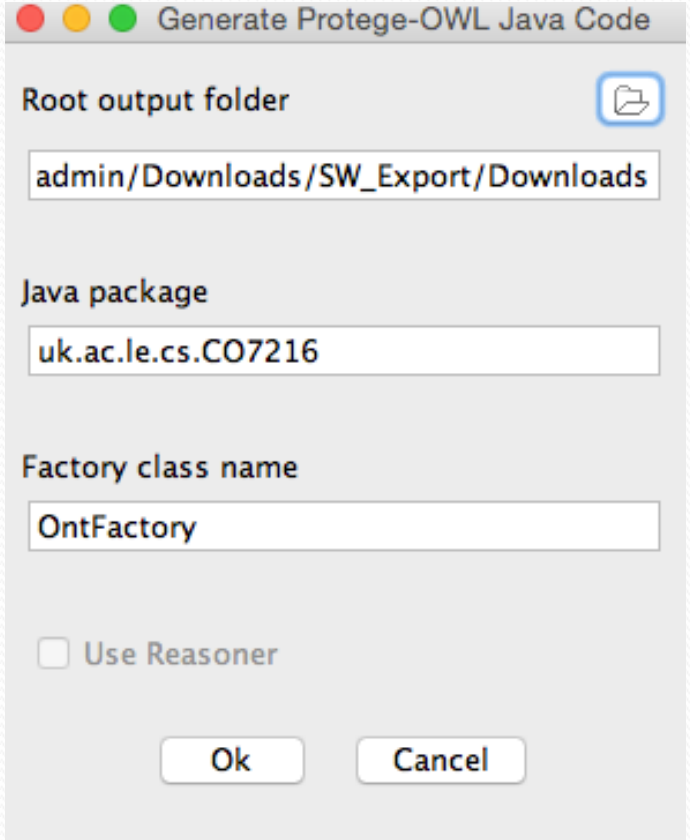
```
String owlFile="file:///Users/ontology/University.owl";

String rules=
    "[knowRule:(?a uol:teach ?module)(?b uol:study ?module)->(?a uol:knows ?b)]";

String sparql="PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>\n"+
"PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n"+
"PREFIX uol:<http://www.cs.le.ac.uk/rdf#>\n"+
"PREFIX owl:<http://www.w3.org/2002/07/owl#>\n"+
"SELECT ?x ?y WHERE {?x uol:knows ?y.}";
.....

Reasoner reasoner =
new GenericRuleReasoner(Rule.parseRules(rules));
   InfModel infmodel = ModelFactory.createInfModel(reasoner, model);
   QueryExecution qe =
          QueryExecutionFactory.create(sparql, Syntax.syntaxARQ, infmodel);
   results = qe.execSelect();
....
```

Part 2: Protégé OWL Code

- Protégé4 provides a Java code generator that can generate Protégé-OWL Java code template based on OWL ontology.
- The Java template allows us to
  - Edit classes/properties in the ontology
  - Create/edit/delete instances.
  - Add/set property values.
  - Save ontology.

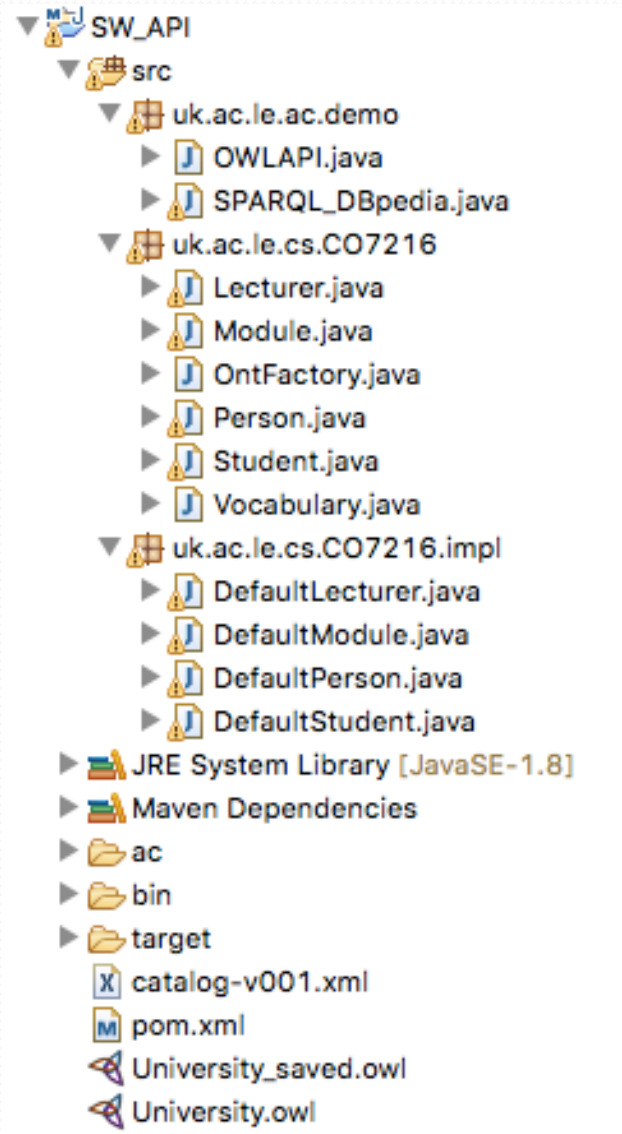
Part 2: Auto-generated Protégé-OWL Java code

- Open University.owl with Protégé 5
- Tools->Generate Protégé-OWL Java code



Part 2: Import generated Java code to the project

- Create a Java project SW_API
- Convert the project to a maven project (Configure->convert to Maven)
- Edit pom.xml (download from blackboard)
- Update maven project (Maven->Update project)
- Import all auto-generated Java files. The project should look like this:



Part 2: Create instances (1)

```
import org.semanticweb.owlapi.apibinding.OWLManager;
import org.semanticweb.owlapi.model.IRI;
import org.semanticweb.owlapi.model.OWLOntology;
import org.semanticweb.owlapi.model.OWLOntologyManager;

import uk.ac.le.cs.CO7216.Lecturer;
import uk.ac.le.cs.CO7216.Module;
import uk.ac.le.cs.CO7216.OntFactory;
import uk.ac.le.cs.CO7216.Student;

....
OWLOntologyManager manager;
        OWLOntology ont;
         public static final String ontology="University.owl";
```

Part 2: Create instances (2)

```
manager = OWLManager.createOWLOntologyManager();

File ontFile=new File(ontology);
 IRI iri = IRI.create(ontFile);
 ont= manager.loadOntologyFromOntologyDocument(iri);
 System.out.println("Loaded ontology: " + ont);

 OntFactory factory=new OntFactory (ont);
 String prefix="http://www.cs.le.ac.uk/rdf#";

 //create an instances of the class Student
 Student s4=factory.createStudent(prefix+"s4");
   //add DatatypeProperty values
            s4.addHas_full_name("Linda");
            s4.addHas_age(new Integer(25));
            s4.addHas_email("linda@student.le.ac.uk");
```

Part 2: Create instances (3)

```
//create an instances of the class Lecturer
        Lecturer l4=factory.createLecturer(prefix+"lecture4");
          //add DatatypeProperty values
                l4.addHas_full_name("Thomas");
                s4.addHas_age(new Integer(35));
                s4.addHas_email("thomas@le.ac.uk");

//create an instance of the class Module
        Module co7216=factory.createModule(prefix+"CO7216");
                    //set ObjectProperty "hasFriend"
                l4.addHasFriend(s4);
                      //set ObjectProperty "teach"
                l4.addTeach(co7216);
                      //set ObjectProperty "study"
                s4.addStudy(co7216);
```

Part 2: Get named individuals

```
//get an existing individual http://www.cs.le.ac.uk/rdf#s4

Student someone=factory.getStudent(prefix+"s4");

//get all modules he/she studies

        for(Module m: someone.getStudy()){
            System.out.println(m.getOwlIndividual());
        }
```

Part 2: Save Ontology

- In-memory OWL model is not saved until saveOntology is called.

```
String ontology_save="University_saved.owl";
save(ontology_save);

....
public void save(String filepath){
                    try{
                    File file=new File(filepath);
                     manager.saveOntology(ont, IRI.create(file.toURI()))
                     System.out.println("Saved ontology: " + ont);
                    }catch(Exception ex){
                              ex.printStackTrace();
                    }
}
```

Reading

- Apache Jena. https://jena.apache.org/index.html
  - ARQ: Query your RDF data
  - Ontology API: Work with models, RDFS and OWL 1
  - Jena TDB: RDF triplestore
  - Fuseki: SPARQL end-point accessible over HTTP

Reading

- Protégé-OWL API
  - Java API and reference implementation for creating, manipulating and serialising OWL Ontologies https://github.com/owlcs/owlapi/wiki/Documentation
- OWL API
  - Open-source Java library for OWL and RDF(S)
  - For the development of components executed inside Protégé-OWL editor (Version 3.X)
  - For development for standalone applications
- Tutorial:
  - http://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide