AMPER Course

RDF(S) Management APIs

Raúl García-Castro, Óscar Corcho, Óscar Muñoz-García
{rgarcia, ocorcho, omunoz}@fi.upm.es
http://www.oeg-upm.net/

Ontology Engineering Group
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain

Index

- **RDF(S) APIs**
- The Jena API
- Hands-on

Sample RDF APIs

RDF libraries for different languages:

- Java, Python, C, C++, C#, .Net, Javascript, Tcl/Tk, PHP, Lisp, Obj-C, Prolog, Perl, Ruby, Haskell
- List in http://esw.w3.org/topic/SemanticWebTools

Usually related to a RDF repository

- **Multilanguage**:
  - Redland RDF Application Framework (C, Perl, PHP, Python and Ruby): http://www.redland.opensource.ac.uk/
- **Java**:
  - Jena: http://jena.sourceforge.net/
  - Sesame: http://www.openrdf.org/
- **PHP**:
  - RAP - RDF API for PHP: http://www4.wiwiss.fu-berlin.de/bizer/rdfapi/
- **Python**:
  - RDFLib: http://rdflib.net/
  - Pyrple: http://infomesh.net/pyrple/

Jena

- Java framework for building Semantic Web applications
- Open source software from HP Labs:
- The Jena framework includes:
  - A RDF API
  - An OWL API
  - Reading and writing RDF in RDF/XML, N3 and N-Triples
  - In-memory and persistent storage
  - A rule based inference engine
  - SPARQL query engine

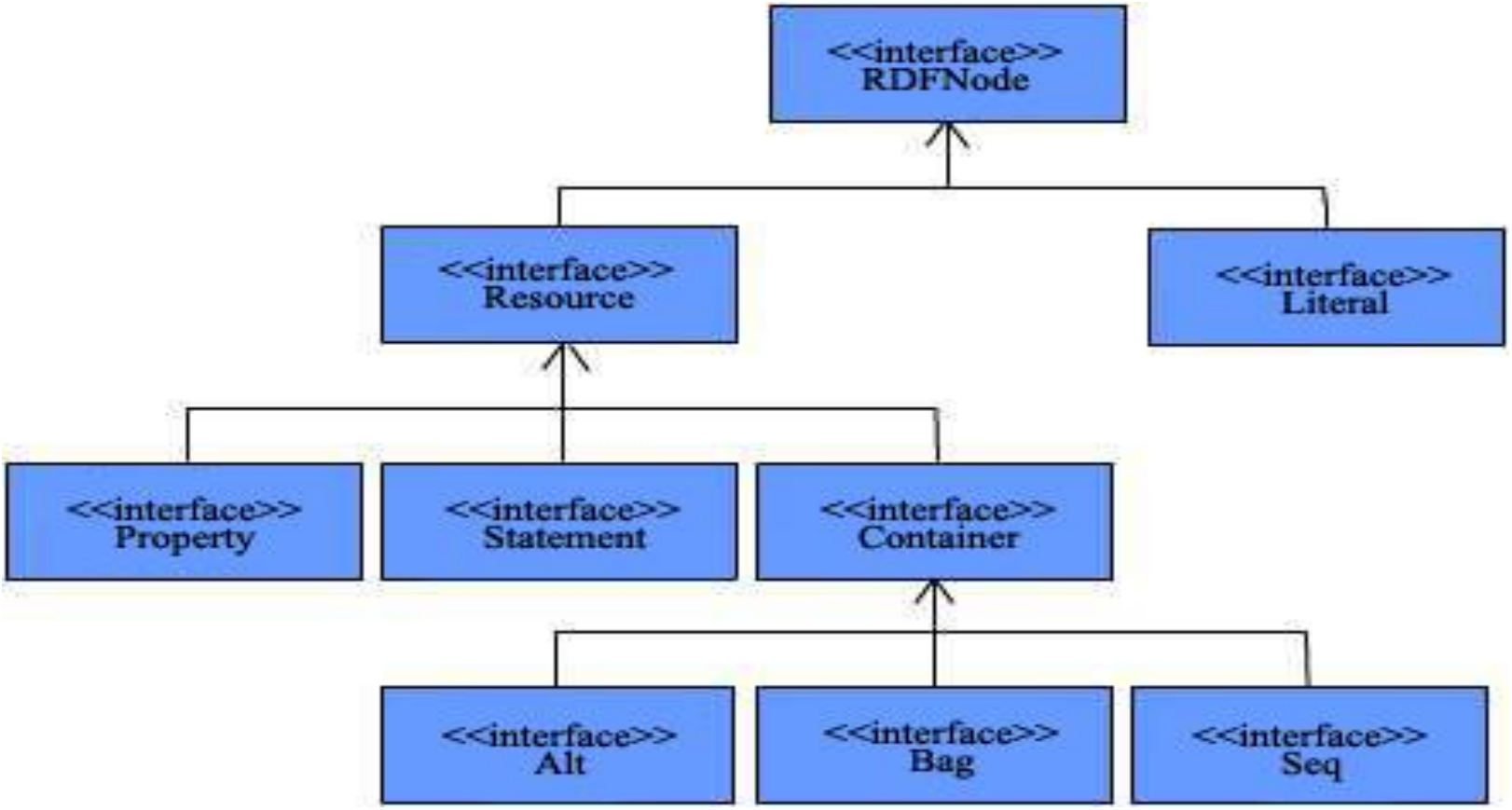
Sesame

- A framework for *storage*, *querying* and *inferencing* of RDF and RDF Schema
- A Java Library for handling RDF
- A Database Server for (remote) access to *repositories* of RDF data
- Highly expressive query and transformation languages
  - SeRQL, SPARQL
- Various backends
  - Native Store
  - RDBMS (MySQL, Oracle 10, DB2, PostgreSQL)
  - main memory
- Reasoning support
  - RDF Schema reasoner
  - OWL DLP (OWLIM)
  - domain reasoning (custom rule engine)

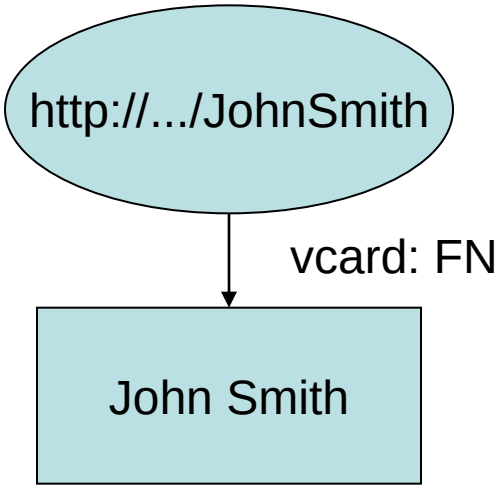
Index

- RDF(S) APIs
- **The Jena API**
- Hands-on

Jena API Structure

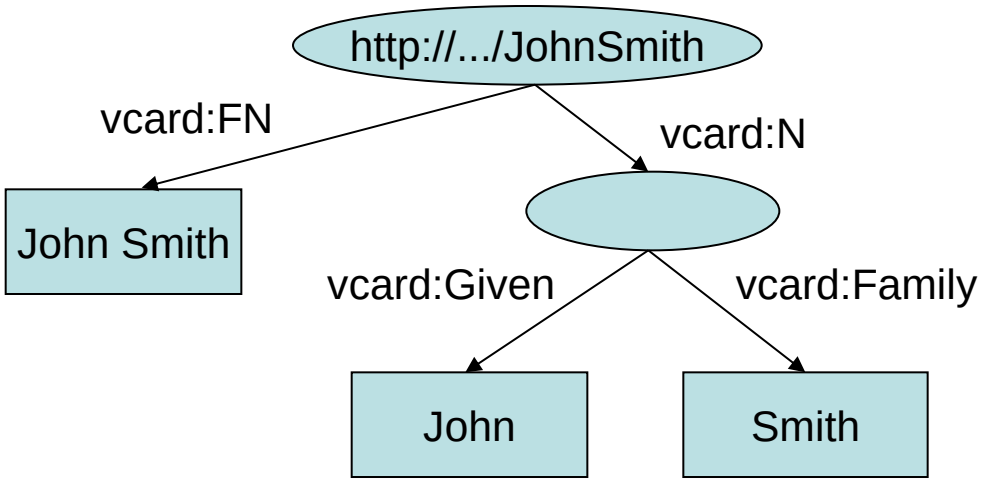


Data model




```
// some definitions
static String personURI = "http://somewhere/JohnSmith";
static String fullName = "John Smith";
// create an empty Model
Model Model model = ModelFactory.createDefaultModel();
// create the resource
Resource johnSmith = model.createResource(personURI);
// add the property
johnSmith.addProperty(VCARD.FN, fullName);
```

Another data 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));
```

Statements

```
// list the statements in the Model
StmtIterator iter = model.listStatements();
// print out the predicate, subject and object of each statement
while (iter.hasNext())
{
    Statement stmt = iter.nextStatement(); // get next statement
    Resource subject = stmt.getSubject(); // get the subject
    Property predicate = stmt.getPredicate(); // get the predicate
    RDFNode object = stmt.getObject(); // get the object
    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
        System.out.print(object.toString());
     }
     else { // object is a literal
        System.out.print(" \"" + object.toString() + "\"");
     }
     System.out.println(" .");
} // end of while
```

```
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#N anon:14df86:ecc3dee17b:-7fff

anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Family "Smith"

anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Given "John"

http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#FN
    "John Smith" .
```

Writing RDF

```
Model model = ModelFactory.createDefaultModel();
Resource jsmith =
model.createResource("http://somewhere/johnsmith")
     .addProperty(VCARD.FN, "John Smith")
     .addProperty(VCARD.N, model.createResource()
     .addProperty(VCARD.Given,"John")
     .addProperty(VCARD.Family, "Smith"));
model.write(new PrintWriter(System.out));
```

```
model.write(System.out, "RDF/XML-ABBREV");

model.write(System.out, "N-TRIPLE");
```

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
 >
  <rdf:Description rdf:nodeID='A0'>
    <vcard:Given>John</vcard:Given>
    <vcard:Family>Smith</vcard:Family>
  </rdf:Description>
  <rdf:Description rdf:about='http://somewhere/johnsmith'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID='A0'/>
  </rdf:Description>
</rdf:RDF>
```

Reading RDF

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

// use the FileManager to find the input file
InputStream in = FileManager.get().open( inputFileName );
if (in == null) {
    throw new IllegalArgumentException("File not found");
}

// read the RDF/XML file
model.read(in, "");

// write it to standard out
model.write(System.out);
```

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
 >
  <rdf:Description rdf:nodeID="A0">
    <vcard:Family>Smith</vcard:Family>
    <vcard:Given>John</vcard:Given>
  </rdf:Description>
  <rdf:Description rdf:about='http://somewhere/JohnSmith/'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID="A0"/>
  </rdf:Description>
...
</rdf:RDF>
```

Navigating a model

```
// retrieve the John Smith vcard resource from the model
Resource vcard = model.getResource(johnSmithURI);
```

Three ways of retrieving property values:

```
// retrieve the value of the N property
Resource name = (Resource) vcard.getProperty(VCARD.N).getObject();

// retrieve the value of the N property
Resource name = vcard.getProperty(VCARD.N).getResource();

// retrieve the given name property
String fullName = vcard.getProperty(VCARD.N).getString();
```

Multiple values in properties

```
// add two nickname properties to vcard
vcard.addProperty(VCARD.NICKNAME, "Smithy")
     .addProperty(VCARD.NICKNAME, "Adman");

// set up the output
System.out.println("The nicknames of \"" + fullName + "\" are:");

// list the nicknames
StmtIterator iter = vcard.listProperties(VCARD.NICKNAME);
while (iter.hasNext()) {
  System.out.println(" " + iter.nextStatement() .getObject() .toString());
}
```

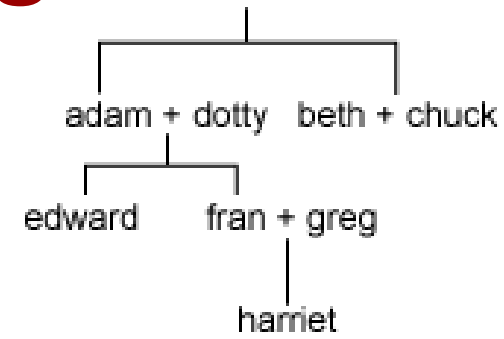
Querying a model

```
// select all the resources with a VCARD.FN property
ResIterator iter = model.listSubjectsWithProperty(VCARD.FN);
if (iter.hasNext()) {
  System.out.println("The database contains vcards for:");
  while (iter.hasNext()) {
    System.out.println(" " + iter.nextStatement()
                                 .getProperty(VCARD.FN) .getString());
  }
} else {
  System.out.println("No vcards were found in the database");
}
```

```
The database contains vcards for:
 Sarah Jones
 John Smith
 Matt Jones
 Becky Smith
```

Create resources

```
// URI declarations
String familyUri = "http://family/";
String relationshipUri = "http://purl.org/vocab/relationship/";

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

// Create a Resource for each family member, identified by their URI
Resource adam = model.createResource(familyUri+"adam");
Resource beth = model.createResource(familyUri+"beth");
Resource dotty = model.createResource(familyUri+"dotty");
// and so on for other family members

// Create properties for the different types of relationship to represent
Property childOf = model.createProperty(relationshipUri,"childOf");
Property parentOf = model.createProperty(relationshipUri,"parentOf");
Property siblingOf = model.createProperty(relationshipUri,"siblingOf");
Property spouseOf = model.createProperty(relationshipUri,"spouseOf");

// Add properties to adam describing relationships to other family members
adam.addProperty(siblingOf,beth);
adam.addProperty(spouseOf,dotty);
adam.addProperty(parentOf,edward);

// Can also create statements directly . . .
Statement statement = model.createStatement(adam,parentOf,fran);

// but remember to add the created statement to the model
model.add(statement);
```



Querying a model

```
// List everyone in the model who has a child:
ResIterator parents = model.listSubjectsWithProperty(parentOf);

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

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

  // Print the URI of the resource
  System.out.println(person.getURI());
}

// Can also find all the parents by getting the objects of all "childOf" statements
// Objects of statements could be Resources or literals, so the Iterator returned
// contains RDFNodes
NodeIterator moreParents = model.listObjectsOfProperty(childOf);

// To find all the siblings of a specific person, the model itself can be queried
NodeIterator siblings = model.listObjectsOfProperty(edward, siblingOf);

// But it's more elegant to ask the Resource directly
// This method yields an iterator over Statements
StmtIterator moreSiblings = edward.listProperties(siblingOf);
```

Using selectors to query a model

```
// Find the exact statement "adam is a spouse of dotty"
model.listStatements(adam,spouseOf,dotty);

// Find all statements with adam as the subject and dotty as the object
model.listStatements(adam,null,dotty);

// Find any statements made about adam
model.listStatements(adam,null,null);

// Find any statement with the siblingOf property
model.listStatements(null,siblingOf,null);
```

Index

- RDF(S) APIs
- The Jena API
- **Hands-on**

Hands-on

- To read an ontology in RDF(S) from two files:
  - GP_Santiago.rdf (conceptualization)
  - GP_Santiago.rdfs (instances)
- To write the class hierarchy of the ontology, including the instances of each class:

```
Class Practica2:MedioTransporte
  Class Practica2:Tren
  Class Practica2:Bicicleta
    Instance Practica2:GP_Santiago_Instance_70
  Class Practica2:Automovil
  Class Practica2:AutoBus
  Class Practica2:APie
Class Practica2:InfraEstructuraTransporte
  Class Practica2:ViaFerrea
  Class Practica2:Sendero
  Class Practica2:Carretera
    Instance Practica2:A6
...
```

Set up

- Requirements:
  - Java JDK 5
  - Eclipse (optional)

- Create a directory for your project
- Install Jena from the USB:
  - Unzip *Jena-2.5.5.zip/**lib*** in the project directory
- Copy the ontologies from the USB:
  - Copy *ontologies/**rdf*** in the project directory

Or copy the *JenaProjectTemplate* directory in your computer

- In Eclipse:
  - Create a new Java project (from existing source)
  - Append the Jena libraries to your classpath if needed (check JDK libs)
  - Write Java code using the Jena API
    http://jena.sourceforge.net/javadoc/index.html
  - Compile
  - Run

Hints

- Create ontology model:

```
public static OntModel
  createOntologyModel(OntModelSpec spec)
```

- Read the ontology in the file

```
Model read(java.lang.String url)
```

- Add all the statements in another model to this model

```
Model add(Model m)
```

More hints

- List root classes

  ```
  ExtendedIterator listHierarchyRootClasses()
  ```

- List subclasses of a class

  ```
  ExtendedIterator listSubClasses(boolean direct)
  ```

- List instances of a class

  ```
  ExtendedIterator listInstances(boolean direct)
  ```

References

- RDF Model and Syntax Specification
  - http://www.w3.org/RDF/
- Jena web site
  - http://jena.sourceforge.net/
- Jena API
  - http://jena.sourceforge.net/tutorial/RDF_API/
- Jena tutorials
  - http://www.ibm.com/developerworks/xml/library/j-jena/index.html
  - http://www.xml.com/pub/a/2001/05/23/jena.html

AMPER Course

RDF(S) Management APIs

Raúl García-Castro, Óscar Corcho, Óscar Muñoz-García
{rgarcia, ocorcho, omunoz}@fi.upm.es
http://www.oeg-upm.net/

Ontology Engineering Group
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain