Developing Smart Java™ Code with Semantic Web Technology

Holger Knublauch
TopQuadrant, Inc.
holger@topquadrant.com
Semantic Web Elevator Pitch

Linked Smart Data

DBpedia model

Conference model

JavaOne model

Geography model

Linked Smart Code

Application-specific code

Generic code
Overview of this talk

> Introduction to RDF
  • Resources, literals and triples
> Linked Data
> RDF Schema
  • Classes, properties, and instances
> Java APIs and Tools
> Data Binding and software architecture
> SPARQL
  • Queries and rules
> Model-driven applications
Semantic Web Technology

> Infrastructure to link data on the Web
> Global database
> Borrows some ideas of object-oriented languages
> Data is self-describing
> W3C standards
  • RDF
  • RDF Schema and OWL
  • SPARQL
  • RDFa, GRDDL
RDF: Resources

> Semantic Web entities have a unique identifier

http://examples.topquadrant.com/conferences/javaone#ScottMcNealy

Namespace

Local name

abbreviated as

javaone:ScottMcNealy
RDF: Triples

Subject | Predicate | Object
--- | --- | ---
javaone:ScottMcNealy | conf:employer | javaone:Sun_Microsystems
RDF: Graphs

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>javaone:ScottMcNealy</td>
<td>conf:employer</td>
<td>javaone:Sun_Microsystems</td>
</tr>
<tr>
<td>javaone:John_Rose</td>
<td>conf:employer</td>
<td>javaone:Sun_Microsystems</td>
</tr>
<tr>
<td>javaone:John_Rose</td>
<td>conf:presenterOf</td>
<td>javaone:JSR_292_Cookbook</td>
</tr>
</tbody>
</table>
RDF: Datatype property values

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>javaone:ScottMcNealy</td>
<td>conf:firstName</td>
<td>“Scott”^^xsd:string</td>
</tr>
<tr>
<td>javaone:ScottMcNealy</td>
<td>conf:lastName</td>
<td>“McNealy”^^xsd:string</td>
</tr>
<tr>
<td>javaone:ScottMcNealy</td>
<td>rdfs:label</td>
<td>“Scott McNealy”</td>
</tr>
</tbody>
</table>
Linked Data

```
javaone:ScottMcNealy

conf:employer

javaone:Sun_Microsystems

owl:sameAs

<http://dbpedia.org/resource/Sun_Microsystems>
```

Sun Microsystems

Sun Microsystems, Inc. (NASDAQ: JAVA) is a multinational vendor of computers, computer components, computer software, and information technology services, founded on February 24, 1982. The company is headquartered in Santa Clara, California (part of Silicon Valley), on the former west campus of the Agnews Developmental Center.

Products include computer servers and workstations based on its own SPARC processors as well as AMD's Opteron and Intel's Xeon processors; storage systems; and, a suite of software products including the Solaris Operating System, developer tools, Web infrastructure software, and identity management applications. Other technologies of note include the Java platform, MySQL and NFS.

Sun is a proponent of open systems in general and Unix in particular, and a major contributor to open source software.

On April 20, 2009, Sun and Oracle Corporation announced that they entered into a definitive agreement under which Oracle will acquire Sun for $7.4 billion.

Sun's manufacturing facilities are located in Hillsboro, Oregon, USA and Lintighow, Scotland.
### About: Sun Microsystems

An Entity in Data Space: dbpedia.org

Sun Microsystems, Inc. Cite error: Invalid `<ref>` tag; refs with no name must have content is a multinational vendor of computers, computer components, computer software, and information technology services. Founded on 24 February 1982. The company is headquartered in Santa Clara, California, on the former west campus of the Agnew Developmental Center. Products include computer servers and workstations based on its own SPARC processors as well as AMD's Opteron and Intel's Xeon processors; storage systems; and a suite of software products including the Solaris Operating System, developer tools, Web infrastructure software, and identity management applications. Other technologies of note include the Java platform and NFS.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia-owl:found创立date</td>
<td>1982</td>
</tr>
<tr>
<td>dbpedia-owl:locationcity</td>
<td>dbpedia:Santa_Clara%2C_California</td>
</tr>
<tr>
<td>dbpedia-owl:locationcountry</td>
<td>dbpedia:United_States</td>
</tr>
<tr>
<td>dbpedia-owl:netincome</td>
<td>403000000</td>
</tr>
<tr>
<td>dbpedia-owl:numberOfEmployees</td>
<td>33350</td>
</tr>
</tbody>
</table>
Linked Data: more RDF triples, for free!
Linked Data: join the cloud
Linked Data: Geography ontology
RDF Schema: Type relationships

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>javaone:ScottMcNealy</td>
<td>rdf:type</td>
<td>conf:Attendee</td>
</tr>
</tbody>
</table>
### RDF Schema: Subclass relationships

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>javaone:ScottMcNealy</td>
<td>rdf:type</td>
<td>conf:Attendee</td>
</tr>
<tr>
<td>conf:Attendee</td>
<td>rdfs:subClassOf</td>
<td>conf:Person</td>
</tr>
</tbody>
</table>
RDF Schema: Class Hierarchy
RDF Schema: Property Domains
RDF Schema: Property Ranges
RDF Schema: from a UML point of view
RDF Schema: linking namespaces
General Java APIs for RDF

> Jena
  • http://jena.sourceforge.net/

> Sesame
  • http://www.openrdf.org/
Jena API: Core interfaces (simplified)

- **Resource**
  - `uri: String`

- **Literal**
  - `lexicalForm: String`
  - `datatype: RDFDatatype`

- **Property**

- **Statement**
  - `subject`
  - `predicate`

- **RDFNode**
  - `object`

- **Model**
  - `

The diagram illustrates the relationships between the core interfaces of the Jena API, showing how resources, literals, properties, statements, and models are interconnected.
Jena API: “Hello, World”

// Create a Jena Model and init some namespace prefixes
String NS = "http://examples.topquadrant.com/conferences/conf#";
Model model = ModelFactory.createDefaultModel();
model.setNsPrefix("conf", NS);
model.setNsPrefix("rdfs", RDFS.getURI());
model.setNsPrefix("xsd", XSD.getURI());

// Create an example triple
Resource javaOne = model.createResource(NS + "JavaOne2009");
Literal label = model.createTypedLiteral("JavaOne 2009");
model.add(javaOne, RDFS.label, label);

// Write the Model in N3 format to the screen
model.write(System.out, FileUtils.langN3);
Jena API: “Hello, World” Output

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix conf: <http://examples.topquadrant.com/conferences/conf#> .

conf:JavaOne2009

   rdfs:label "JavaOne 2009"^^xsd:string .

RDF file in N3 format
Options include RDF/XML
Tools: OWL ontology editor Protege
Tools: TopBraid Composer
Development Methodology

> Define/re-use schema
> Play with example instances, queries etc
> Take a data-centric approach!
> Link key classes of the schema to your Java code (Data Binding)
> Assume schema evolution
> Develop smart, generic code instead of domain-specific code
Data Binding: Generated schema class

```java
package com.topquadrant.examples.conf.vocabulary;

import com.hp.hpl.jena.rdf.model.Property;

/**
 * Vocabulary for http://examples.topquadrant.com/conferences/conf
 */

class CONF {

    public final static String BASE_URI = "http://examples.topquadrant.com/conferences/conf";
    public final static String NS = BASE_URI + "#";
    public final static String PREFIX = "conf";

    public final static Resource Address = ResourceFactory.createResource(NS + "Address");

    public final static Resource Attendee = ResourceFactory.createResource(NS + "Attendee");

    public final static Resource Company = ResourceFactory.createResource(NS + "Company");

    public final static Resource Conference = ResourceFactory.createResource(NS + "Conference");
```
Data Binding: Basic Architecture

- **RDFNode**
  - **Resource**
  - **Literal**
    - **Person**
      - getFirstName()
      - getFullName()...

- **Generic Code (SPARQL etc)**
- **Specific Code**
Data Binding: Example use

```java
Model model = ModelFactory.createDefaultModel();

// Create a new Person instance
String steveURI = CONF.ns + "SteveJobs";
Person steve = ConfFactory.createPerson(model, steveURI);

// Set some property values
steve.setFirstName("Steve");
steve.setLastName("Jobs");

// Query some property values
String fullName = steve.getFullName();
System.out.println("Person: " + fullName);
```
public interface Person extends Resource {

    String getFirstName();
    String getFullName();
    String getLastName();
    void setFirstName(String value);
    void setLastName(String value);
}

Data Binding: Example interface
Data Binding: Example implementation

```java
public class PersonImpl extends ResourceImpl implements Person {

    public PersonImpl(Node node, EnhGraph eg) {
        super(node, eg);
    }

    public String getFirstName() {
        return getStringProperty(CONF.firstName);
    }

    public String getFullName() {
        return getFirstName() + " " + getLastName();
    }
}
```
Data Binding: Factory class

```java
public class ConfFactory {
    static {
        register(Person.class, PersonImpl.class);
    }

    public static Person createPerson(Model model, String uri) {
        return model.createResource(uri, CONF.Person).as(Person.class);
    }

    public static Person getPerson(Model model, String uri) {
        return model.createResource(uri).as(Person.class);
    }

    ...
```
Data Binding: Frameworks

> ELMO
  • http://www.openrdf.org/doc/elmo/1.4/

> Sommer
  • https://sommer.dev.java.net/sommer/index.html

> JenaBean
  • http://code.google.com/p/jenabeen/
  • http://www.incunabulum.de/projects/it/owl2java/

> Kazuki
  • http://projects.semwebcentral.org/projects/kazuki/
Data Binding: Java code generators

> RDF Reactor
  • http://semanticweb.org/wiki/RDFReactor

> ELMO
  • http://www.openrdf.org/doc/elmo/1.4/

> Jastor
  • http://jastor.sourceforge.net/

> OWL2Java
Intermediate Summary

> Data model in RDF and RDF Schema
> Data binding to link RDF and Java worlds

> Next: let the data work for us with SPARQL
SPARQL: Example Query

```
SELECT ?person ?company
WHERE {
}
```
SPARQL: Query results as subgraphs
SPARQL: Example 2

```
SELECT ?company ?label
WHERE {
}
```
SPARQL: Example 3

```
SELECT ?company (COUNT(?presentation) AS ?count)
WHERE {
  ?person conf:presenterOf ?presentation
} GROUP BY ?company
```

<table>
<thead>
<tr>
<th>company</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>javaone:The_MathWorks</td>
<td>2</td>
</tr>
<tr>
<td>javaone:ThoughtWorks</td>
<td>4</td>
</tr>
<tr>
<td>javaone:TomTom</td>
<td>1</td>
</tr>
<tr>
<td>javaone:TopQuadrant_Inc.</td>
<td>2</td>
</tr>
<tr>
<td>javaone:Trailer_Park</td>
<td>1</td>
</tr>
<tr>
<td>javaone:Training_Consulting</td>
<td>2</td>
</tr>
</tbody>
</table>
Jena API: SPARQL

Model m = ModelFactory.createDefaultModel();
m.read(new FileReader("conf.n3"), null, FileUtils.langN3);
m.read(new FileReader("javaOne.rdf"), null);
String queryString =
"PREFIX " + CONF.PREFIX + " : <" + CONF.NS + " > \n" +
"SELECT ?person ?company \n" +
"WHERE { \n" +
"  ?person conf:employer ?company . \n" +
"}"
;
Query query = QueryFactory.create(queryString);
QueryExecution qx = QueryExecutionFactory.create(query, m);
Jena API: SPARQL (cont’d)

```java
ResultSet rs = qx.execSelect();
while(rs.hasNext()) {
    QuerySolution s = rs.nextSolution();
    Resource person = (Resource) s.get("person");
    Resource company = (Resource) s.get("company");
    System.out.println(person.getLocalName() + "\t" +
                        company.getLocalName());
}
```

Console output:

- Angela_Caicedo  Sun_Microsystems_Inc.
- Matthew_Fisher   Progeny_Systems
- Aditya_Dada      Sun_Microsystems_Inc.
- Roy Ben Havun    Sun_Microsystems
SPARQL: Querying the schema

“Which properties are relevant for a given resource?”
SPARQL: Model-driven forms

[Diagram of model-driven forms with a resource form and a property form showing annotations and other properties such as address, city, country, latitude, longitude, and zip code.]
SPARQL: Model-driven applications
SPARQL: CONSTRUCT rule
SPARQL: generalized rule

```
CONSTRUCT {
    ?instance rdf:type ?superClass .
}
WHERE {
    ?subClass rdfs:subClassOf ?superClass .
    ?instance rdf:type ?subClass .
}
```
Adding domain-specific properties; the application’s code knows nothing about them at compile-time.
Rules: Conference Sponsorship

“Companies that are co-sponsors get 5 free conference passes”

**Condition**
(triple pattern in the graph)

**Action**
(triples to add/infer)
Rules: SPARQL Inferencing Notation

Class Form

Name: conf:Company

Annotations
rdfs:label
Company

Class Axioms
rdfs:subClassOf
owl:Thing

spin:rule

CONSTRUCT {
    ?this javaone:freePasses 5 .
} WHERE {
    ?this javaone:cosponsorOf javaone:JavaOne2009 .
}

Other Properties
Example Application (Unit Conversion)
Example Application (Computer Game)

composing-the-semantic-web.blogspot.com
Summary

> New development paradigm based on Linked Data
> Smart code walks the data graph at run-time
> Very little hard-coding of behavior

> Benefits
  > flexible architecture
  > faster turn-around times

> Challenges
  > learning curve (this is not OO!)
  > integration with established technologies
Thank You

Holger Knublauch
holger@topquadrant.com
http://www.topquadrant.com
Backup Slides
RDF Databases

- AllegroGraph
- Jena (RDB, SDB, TDB)
- Mulgara
- Oracle 11g
- Sesame
Inferencing APIs

> Often integrated into RDF stores:
  • Jena Rules
  • Sesame
  • Oracle 11g RDF

> OWLIM (Rule engine with RDFS/OWL support)
  • http://ontotext.com/owlim/

> Pellet (OWL DL)
  • http://clarkparsia.com/pellet/

> TopSPIN (Rules via SPARQL)
  • http://spindrdf.org/api