Semantic Web for Java Developers
19 May 2010

Steve Hamby

VP, Semantic Technologies & Information Fusion

shamby@orbistechnologies.com

tel: 678.346.6386

fax: 404.963.0914

orbis TECHNOLOGIES, INC.

Agenda

- What is Semantic Web
- Semantic Web and IT Evolution
- What Are Issues / Concerns
- Future Perspective
- Basic Terminology
- Java and Semantic Web
- Jena
- Sesame
- Manchester OWL API

What Is the Semantic Web

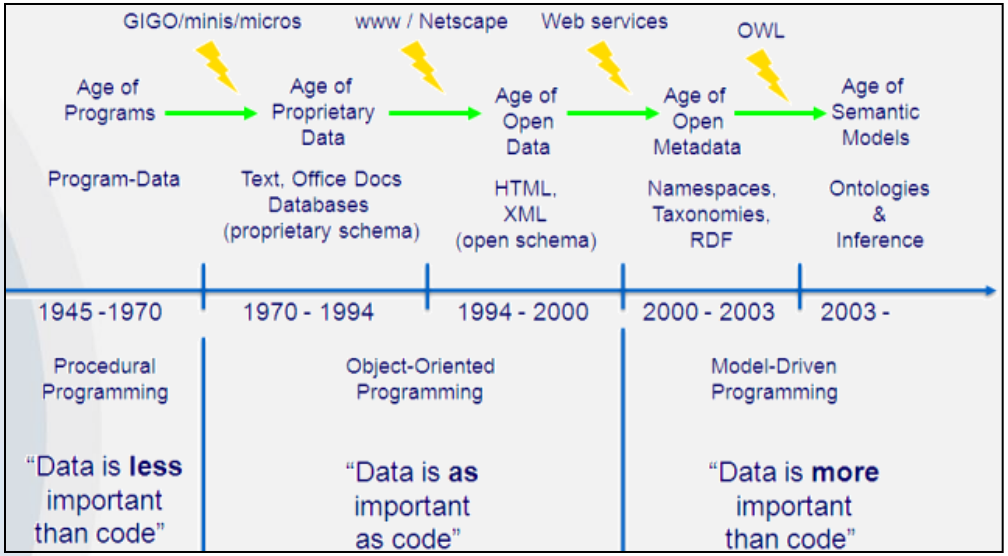
- An evolution of the World Wide Web known as Web 3.0
- A vision created by Tim Berners-Lee (Director, World Wide Web Consortium (W3C))

  *"... a goal of the Web was that, if the interaction between person and hypertext could be so intuitive that the **machine-readable** information space gave an accurate representation of the state of people's thoughts, interactions, and work patterns, then **machine analysis** could become a very powerful management tool, seeing patterns in our work and facilitating our working together through the typical problems which beset the management of large organizations."*

  source: http://www.w3.org/People/Berners-Lee/1996/ppf.html
- A set of standards maintained by W3C to implement the Semantic Web
  - These standards are used by enterprises for non-Semantic-Web uses.
  - The standards provide guidance for interoperability between competing technologies and software, similar to what web services standards provided for that community

Semantic Technologies and IT Evolution: Code-Centric vs. Data-Centric

- Semantic technologies are part of an evolution of IT focus from code to data
  - In the Code Centric years, data was often stored in flat files with no structure , while complex large "edit" programs contained all knowledge about the data
  - The creation of databases, specifically Network and RDBMS was one of the first steps leading to Data Centric evolution
  - The last decade has seen standards such as XML and now RDF/OWL that further evolve IT to a Data Centric environment



Source: Mike Daconta "Information as a Product" http://www.gca.org/xmlusa/2004/slides/daconta/Creating Relevance and Reuse With Targeted Semantics.ppt

How Are Semantic Technologies Used

- Advanced Search
  - Search Enhancement / Query Replacement
  - Faceted Navigation
- Process / Service / Product Optimization
  - Defect Detection
  - Process Modeling and Reengineering
- Semantic Integration
  - Semantic Model of Integrated Data Silos
  - Entity Extraction Model for Integrating Unstructured Text
  - Semantic Fusion Models
- Compliance Applications (Reporting, etc)
  - Gold-Standard (Compliance Regulations, etc) can be Modeled as Ontology, Providing Agile Approach

What Are The Issues / Concerns

- Industry Overselling
  - Several companies are selling as ‘end-to-end’ when they are not, and leaving customers with negativity towards semantic technologies
  - Too many companies trying to learn technologies on the customer $
    - Plenty of room for cost management with open source: GATE NLP, Pellet, Sesame, Jena, etc.
  - Standards-based technology does not equate to interoperable software
    - Requires additional vendor guidelines on interoperability

- Semantic Web Overselling
  - Autonomous but interoperable agents for automated discovery and disambiguation are not currently reality … AFAIK
  - Concepts are gaining ground with C-levels, but are we at the “Slope of Enlightenment” or the “Peak of Inflated Expectations”
    - “Trough of Disillusionment” or “Plateau of Productivity” could be next

- Software / Technology Acquisitions

Issues / Concerns (cont.)

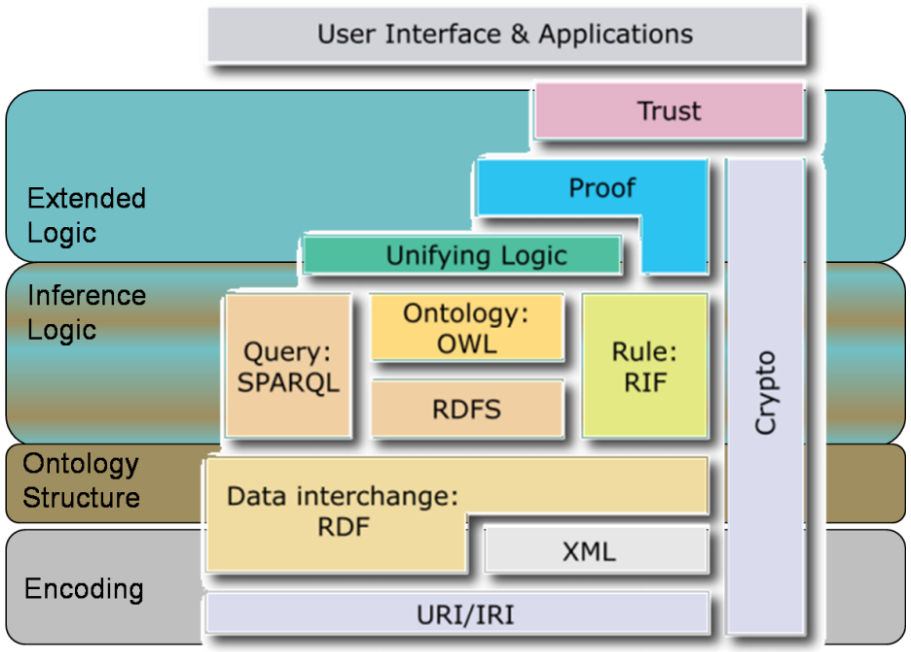
- Lack of enterprise-tested applications
  - Discussions revolve around billions of triples, not pedabytes of data
  - Will require time to enhance existing tools
  - Initial semantic technology applications must be considered an investment
    - Similar to XML and SOA, “semantic plumbing” costs will be recouped on subsequent applications
- Competing standards from ISO and W3C … and others
  - Various levels of “smart data”; different standards needed for each
  - Need to improve standards interoperability
- Semantic technologies are viewed as revolutionary in many IT shops
  - It is an evolution of IT migration that started in the 70s
  - Changing perceptions can assist adoption

What To Expect in Near Future

- More industry “anchors” supporting these standards
  - Oracle Spatial support for RDF and OWL 3+ years ago gave a major boost to the success of this movement
- Similar standards to WS-I … interoperability is MORE important than standards-based
- Intersection of Cloud Computing and Semantic Technologies
  - Already happening, but is growth area that leads to enterprise-ready semantic technology software … and, eventually, the SemWeb vision
- Intersection of Social Web technologies and Semantic Web technologies
  - Already happening: Semantic wikis, etc.
  - Huge potential and impact
- Common Attributes of Successful Implementations
  - Networking to share issues, how-to’s, etc
  - Get your C-level buy-in
  - Drive vendors with your requirements
  - Keep multiple vendors ready to work your problem – heavy M&A activity

Semantic Web Basic Terms

- **Ontology** is a formalized vocabulary of terms with explicitly defined relationships that specify a domain and is shared by a community of users
- **Resource Description Framework (RDF)** builds on XML to provide a simple data model to make statements about resources on the web and leverages URIs/IRIs to identify resources
- **Web Ontology Language (OWL)** is a layered ontology representation language based on Description Logics and RDF using Classes, Datatype and Object Properties, and Individuals with a set of defined relationships – equivalence, restriction, subclass, etc.
- **Rules Interchange Format (RIF)** provides a standard formats for interchange of rules across languages and rule engines
- **SPARQL** is a query language and protocol for RDF



Source: Semantic Stack from W3C…extended by Orbis Technologies, Inc

Fundamental Assumptions in OWL:
Why is this Different

- Open-World vs. Closed-World
  - Huge difference between ontologies and relational databases
  - Closed-World – Any statement not explicitly stated is assumed false
  - Open-World – Doesn't assume an answer is false unless absolutely proven to be false
    - Many questions may simply have no provable answer
  - OWL assumes Open World
- Monotonic vs. Non-Monotonic
  - Monotonic – Adding new statements does not falsify previous conclusion, thereby allowing and requiring consistency checks
  - OWL and DL are monotonic
- Provable vs. Satisfiable
  - OWL supports both proving a statement or determining if possible
  - A fact is provably true if all cases are true
  - A fact is satisfiably true if any case is true

orbis TECHNOLOGIES, INC.

Semantic Technologies and Java

- 1995 Sun article defined Java as

  A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multi-threaded, and dynamic language

  – Semantic Web and Java are both “buzzword compliant” ☺

- Classes can represent real world phenomena

- Hierarchical class structure where subclass inherits properties and methods of superclass

- Strong typing

- Three useful APIs in Semantic Web community are Java APIs:
  – Jena – Java API for building Semantic Web applications
  – Sesame – API from http://openrdf.org
  – Manchester OWL API – API from University of Manchester for manipulating OWL ontology

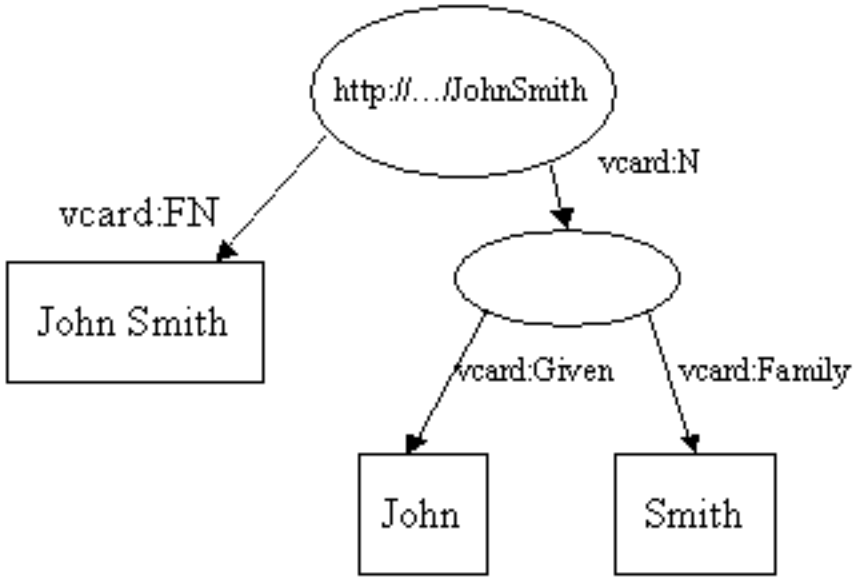
orbis TECHNOLOGIES, INC.

Jena Overview

- Jena is a Java framework for building Semantic Web applications
- Jena is open source software developed in HP Labs
- 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 support
- Two versions:
  - Jena 1
    - Expressive support for RDF
    - Limited reasoning facilities (RDQL)
  - Jena 2
    - Ontology API included
    - Support for OWL included

Jena RDF API

- Allows creating and manipulating RDF Models from Java applications.
- Provides Java classes to represent:
  - Models
  - Resources
  - Properties
  - Literals
  - Statements




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

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

Resource johnSmith = model.createResource(personURI);

johnSmith.addProperty(VCARD.FN, fullName);

johnSmith.addProperty(VCARD.N, model.createResource()
            .addProperty(VCARD.Given, givenName)
            .addProperty(VCARD.Family, familyName));
```

Jena and the RDF Model

- To serialize the model in XML:

```
model.write(System.out);
```

- To load a model in the memory:

```
Model model = ModelFactory.createDefaultModel();
model.read("file:c:/example.owl");
```

- Getting information via the URI of the resource:

```
// retrieve the resource John Smith
String johnSmithURI = "http://somewhere/JohnSmith";
Resource jSmith = model.getResource(johnSmithURI);

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

// retrieve the value of the property FN
String fullName = (String) jSmith.getProperty(VCARD.FN).getObject();
```

Jena and the RDF Model

- Searching information in a model:

```
// retrieve all the resources of the type vcard
// (assuming that all such resources have a property FN)
ResIterator it = model.listSubjectsWithProperty(VCARD.FN);
while (it.hasNext()) {
    Resource r = it.nextResource();
    System.out.println(r);
}
```

- More advanced querying:
  - Use of construction listStatements(Selector s)
  - Use of RDQL

- Support of the following operations:
  - Union
  - Intersection
  - Difference

```
// reading the RDF models
model1.read(new InputStreamReader(in1), "");
model2.read(new InputStreamReader(in2), "");

// unifying RDF models
Model model = model1.union(model2);
```

Jena and RDBMS ←→ RDF

- The package com.hp.hpl.jena.db is used to provide persistent storage of Jena Models
- Accessing a Model in a RDBMS (support for Oracle, MySQL, etc):

```
try {
// Load MySQL driver
        Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException e) { ... }

// Create a database connection
IDBConnection conn =
new DBConnection("jdbc:mysql://localhost/jenadb", “user”, “pass”, "MySQL");

ModelMaker maker = ModelFactory.createModelRDBMaker(conn);
// Retrieve Model
Model dbModel = maker.openModel(“http://www.example.com/example", true);
// View all the statements in the model as triples
StmtIterator iter = dbModel.listStatements();
while(iter.hasNext()) {
        Statement stmt = (Statement)iter.next();
        System.out.println(stmt.asTriple().toString());
}
```

Jena and Ontology API

- Supports RDF Schema, DAML, DAML+OIL and OWL
- Language independent
- Provides Inference Support
- OntModel
  - Contains ontology statements
  - Can be used to retrieve existent resources (Classes, individuals, properties etc) or create new ones
- Classes are represented by OntClass
  - OntClass methods can be used to view the instances, superclasses, subclasses, restrictions etc of a particular class
- OntClass provides methods in order to assert subclass/superclass relations, or class/instance relations
- Classes may be just ‘labels’ under which individuals are categorized, but they can be more complex, e.g. described using other class definitions
  - UnionClass, IntersectionClass, EnumeratedClass, ComplementClass, Restriction
  - The OWL API provides ways to determine whether a class falls on one of the above categories
  - OntModel provides methods to construct such complex definitions

Jena Overview and Ontology Model

- Use of method createOntologyModel()
- Possible to specify:
  - Used language
  - Associated reasoning

```
String fileName = "c:/example.owl";
String baseURI = "file:///" + fileName;

OntModel model =
   ModelFactory.createOntologyModel(ProfileRegistry.OWL_DL_LANG);

model.read(new FileReader(schemaFileName), baseURI);
```

Jena Ontology Classes & Properties

- Classes are basic construction blocks represented as OntClass.

```
OntClass camera = model.getOntClass(camNS + "Camera");
for (Iterator i = camera.listSubClasses(); i.hasNext();) {
  OntClass c = (OntClass) i.next();
  System.out.println(c.getLocalName());
}
```

- Properties are represented via OntProperty

```
OntModel m = ModelFactory.createOntologyModel();
OntClass Camera = m.createClass(camNS + "Camera");
OntClass Body = m.createClass(camNS + "Body");

ObjectProperty part = m.createObjectProperty(camNS + "part");
ObjectProperty body = m.createObjectProperty(camNS + "body");

body.addSuperProperty(part);
body.addDomain(Camera);
body.addRange(Body);
```

Jena and Complex Classes

- It is possible to define classes by means of operations for union, intersection, difference

```
<owl:Class rdf:ID="SLR">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Camera"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#viewFinder"/>
            <owl:hasValue rdf:resource="#ThroughTheLens"/>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

```
// create instance throughTheLens
OntClass Window = m.createClass(camNS + "Window");
Individual throughTheLens =
   m.createIndividual(camNS + "ThroughTheLens", Window);

// create property viewfinder
ObjectProperty viewfinder =
   m.createObjectProperty(camNS + "viewfinder");

// create restriction hasValue
HasValueRestriction viewThroughLens =
   m.createHasValueRestriction(null, viewfinder, throughTheLens);

// create class Camera
OntClass Camera = m.createClass(camNS + "Camera");

// create intersection for defining class SLR
IntersectionClass SLR = m.createIntersectionClass(camNS + "SLR",
        m.createList(new RDFNode[] {viewThroughLens, Camera}));
```

Jena and Managing Instances

- Can define schema
  - Classes
  - Properties (DataTypeProperty, ObjectProperty)
  - Restrictions
    - Types of Data
    - Cardinality
- Can define individuals
- Other individual operations
  - model.listIndividuals()
    - Returns all instances of the model
  - individual.hasProperty(Property p,Object o)
    - Returns True if there is an individual having property p with value o
  - ontClass.listInstances();
    - Returns all instances of the class

```
URI = http://test/camera/#Camera
OntClass c = model.getOntClass(URI +#Camera")
OntProperty p = model.getOntProperty(URI +#name")

Individual ind =  model.getIndividual(URI +#camera1")
if (ind.hasProperty(p))
      Statement st = ind.getProperty(p);
      Object  l = (Object)st.getObject();
```

```
<owl:Class rdf:ID="Camera">
      <rdfs:subClassOf rdf:resource="#Item"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="name">
      <rdfs:domain rdf:resource="#Camera"/>
      <rdfs:range rdf:resource=“xsd:string"/>
</owl:DatatypeProperty>

 <camera:Camera rdf:ID="camera1">
            <camera:name>Kodak</camera:name>
 </camera:Camera>
```

Jena SPARQL Engine

- Jena uses the ARQ engine for the processing of SPARQL queries
  - The 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

Jena SPARQL Engine Example

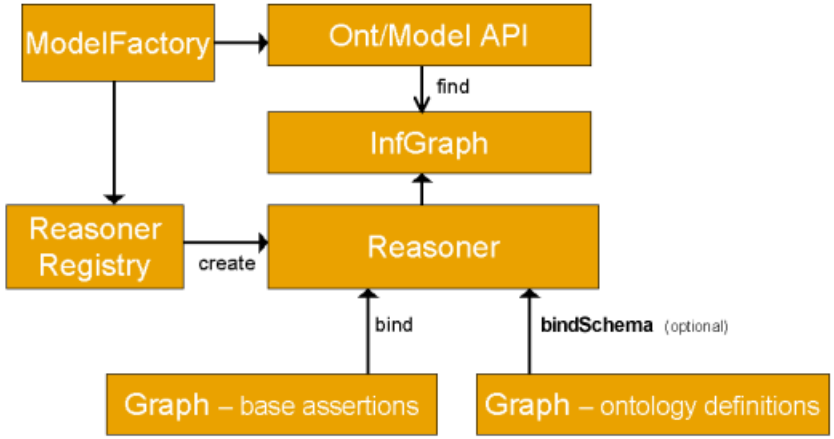
```
// Prepare query string
String queryString =
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
"PREFIX : <http://www.example.com/onto1#>\n" +
"SELECT ?married ?spouse WHERE {" +
"?married rdf:type :MarriedPerson.\n" +
"?married :hasSpouse ?spouse." +
"}";
// Use the ontology model to create a Dataset object
// Note: If no reasoner has been attached to the model, no results
// will be returned (MarriedPerson has no asserted instances)
Dataset dataset = DatasetFactory.create(ontModel);
// Parse query string and create Query object
Query q = QueryFactory.create(queryString);
// Execute query and obtain result set
QueryExecution qexec = QueryExecutionFactory.create(q, dataset);
ResultSet resultSet = qexec.execSelect();
```

Jena SPARQL Engine Example

```
// Print results
while(resultSet.hasNext()) {
        // Each row contains two fields: ‘married’ and ‘spouse’,
        // as defined in the query string
        QuerySolution row = (QuerySolution)resultSet.next();

        RDFNode nextMarried = row.get("married");
        System.out.print(nextMarried.toString());

        System.out.print(" is married to ");

        RDFNode nextSpouse = row.get("spouse");
        System.out.println(nextSpouse.toString());
}
```

Jena Inference Support

- Designed to allow a range of inference engines or reasoners to be plugged into Jena
- Includes a generic rule engine that can be used for many RDF processing or transformation tasks
- Jena comprises a set of basic reasoners
  - OWL Reasoner
  - DAML Reasoner
  - RDF Rule Reasoner
  - Generic Rule Reasoner
  - Reasoner plug-in



Jena Inference Support

Java Code:

```
Model schema = ModelLoader.loadModel("file:c:/Schema.owl");
Model data = ModelLoader.loadModel("file:c:/example.owl");
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
reasoner = reasoner.bindSchema(schema);
InfModel modelInf = ModelFactory.createInfModel(reasoner, data);

ValidityReport vrp1 = modelInf.validate();
if (vrp1.isValid()){
        System.out.println("Valid OWL");
}else {
        System.out.println("Not valid OWL");
        for (Iterator i = vrp1.getReports(); i.hasNext();){
                System.out.println(" - " + i.next());
}
```

Ontology:

```
<owl:Class rdf:ID="Camera">
      <rdfs:subClassOf>
        <owl:Restriction>
                <owl:onProperty rdf:resource="#name" />
            <owl:maxCardinality

      rdf:datatype="xsd:nonNegativeInteger">1</owl:maxCardinality>
        </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
<camera:Camera rdf:ID="camera1">
                <camera:name>KODAK</camera:name>
                <camera:name>OLIMPUS</camera:name>
</camera:Camera>
```

Result:

Error (too many values): Too many values on max-N property (prop, class)
Culprit = http://www.xfront.com/owl/ontologies/camera/#camera1
Implicated node: http://www.xfront.com/owl/ontologies/camera/#name
Implicated node: http://www.xfront.com/owl/ontologies/camera/#Camera

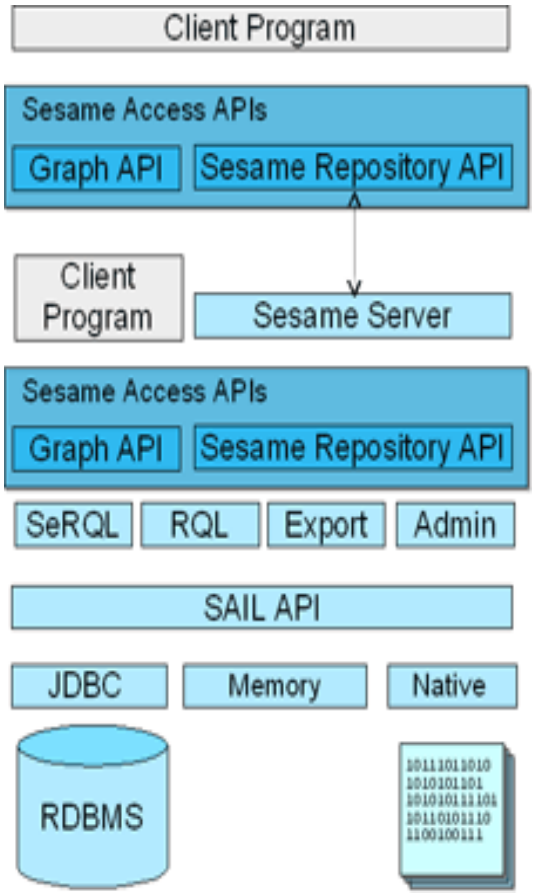
orbis TECHNOLOGIES, INC.

Sesame Overview

- An open source Java framework and API for RDF applications
- Similar in capability and purpose to Jena API
- Mature API (2004) largely adopted by Semantic Web community
- API provides support for common tasks such as storage, SPARQL queries and inferencing of RDF (with support for named graphs)
- Framework can be deployed as web server or as a library
- Provides lower-level RDF storage and inference abstraction layer (SAIL) similar to JDBC abstraction of databases
- Provides higher-level Repository API to connect, add, remove, and query RDF
- Active developer community has provided many extensions including SAILs for Oracle, MySQL, PostgreSQL, Mulgara data stores & RDFS SAIL
- Higher level adaptations include Elmo which directly implements common ontologies such as FOAF, RSS & Dublin Core
- RIO parser provides read/write support for RDF/XML, N-Triples & N3
- Additional libraries have been developed to support PHP, C# & Python development

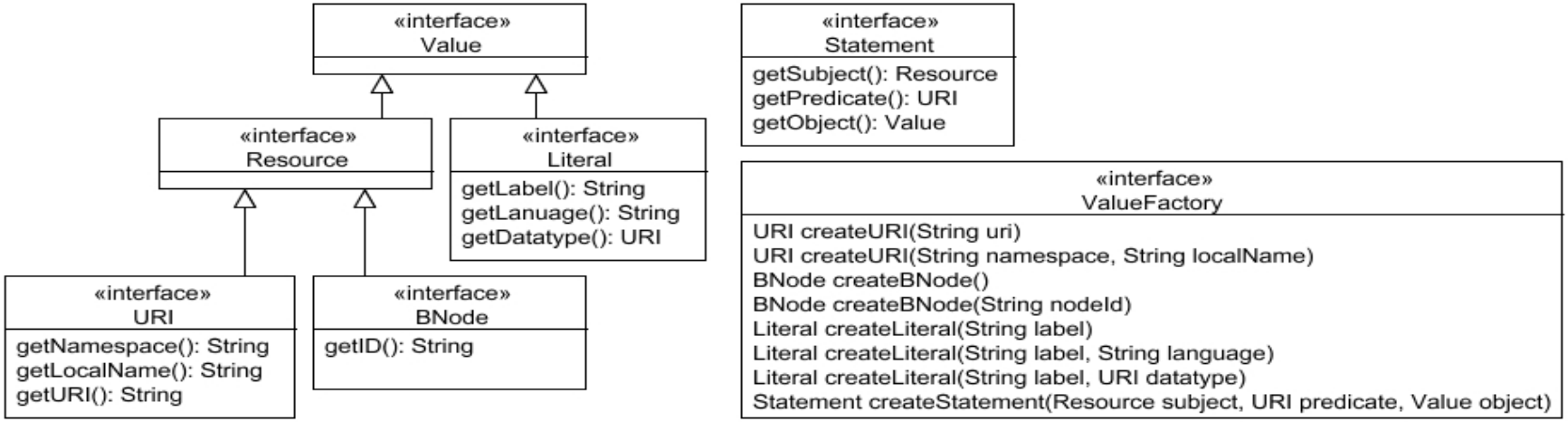
Sesame Overview

- Sesame at its core implements construction and manipulation of RDF Model
- Two main communication interfaces
  - SAIL API (lower-level)
  - Repository API (higher-level)
- Client program connects through Repository API to add/remove/query RDF in underlying SAIL
- SAIL layer provides connection and abstraction to inference engine and memory-based, file-based & database stores
- Repositories can be Local or Remote
- Local Repositories connect directly to store
- Remote Repositories connect via HTTP
- Architecture allows flexible configurations of different data stores and inferencers through a standard API
- Allows flexibility and portability



Sesame API Overview

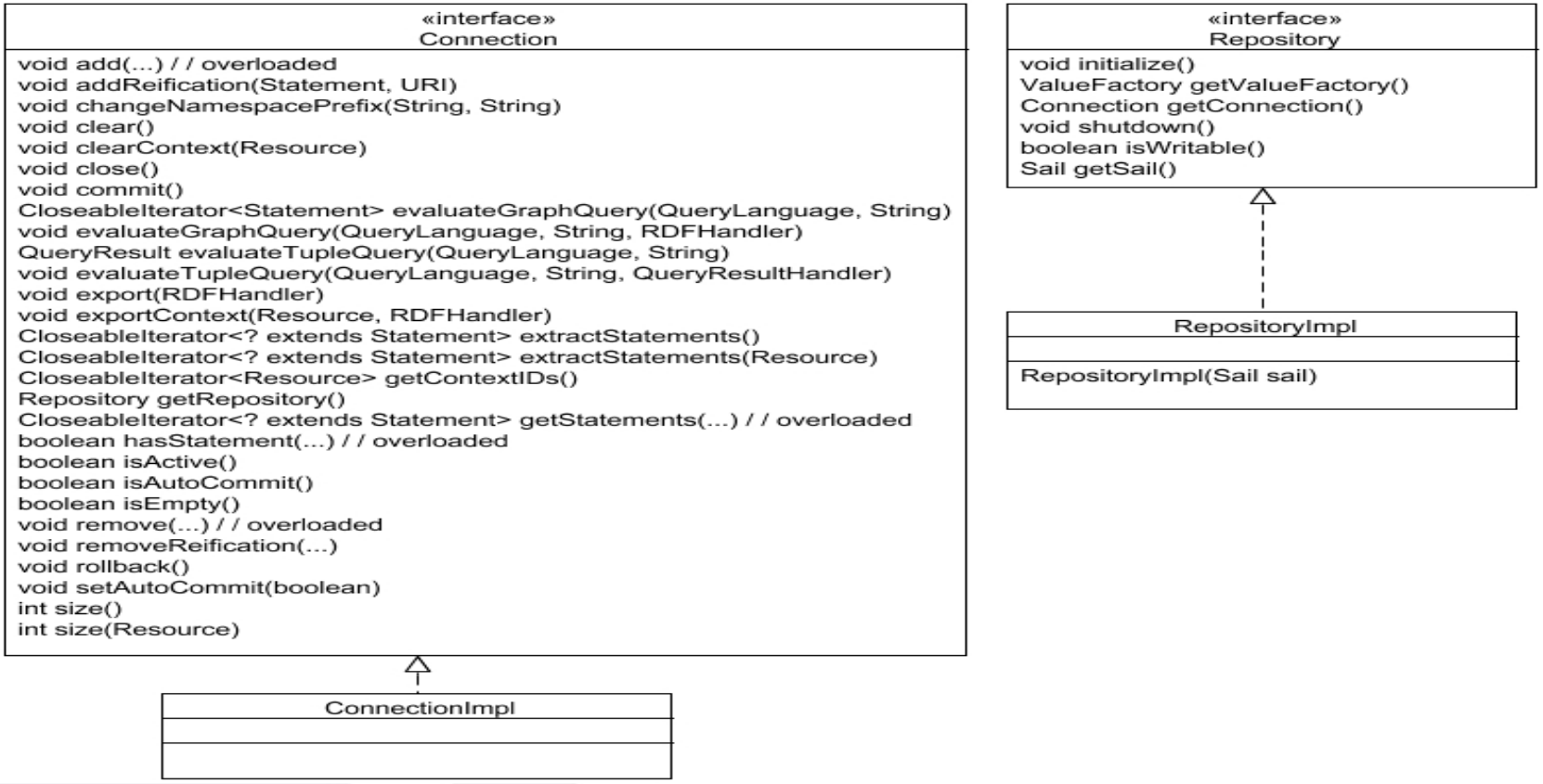
- Sesame library consists of four core jars
  - Sesame.jar: Core Sesame classes
  - rio.jar: (RDF I/O) Parsers to read/write RDF
  - openrdf-model.jar: Core RDF model
  - openrdf-util.jar: Shared utility classes
- Core RDF Model allows creation of URIs, Resources, Literals, Statements



Class diagram for the RDF model

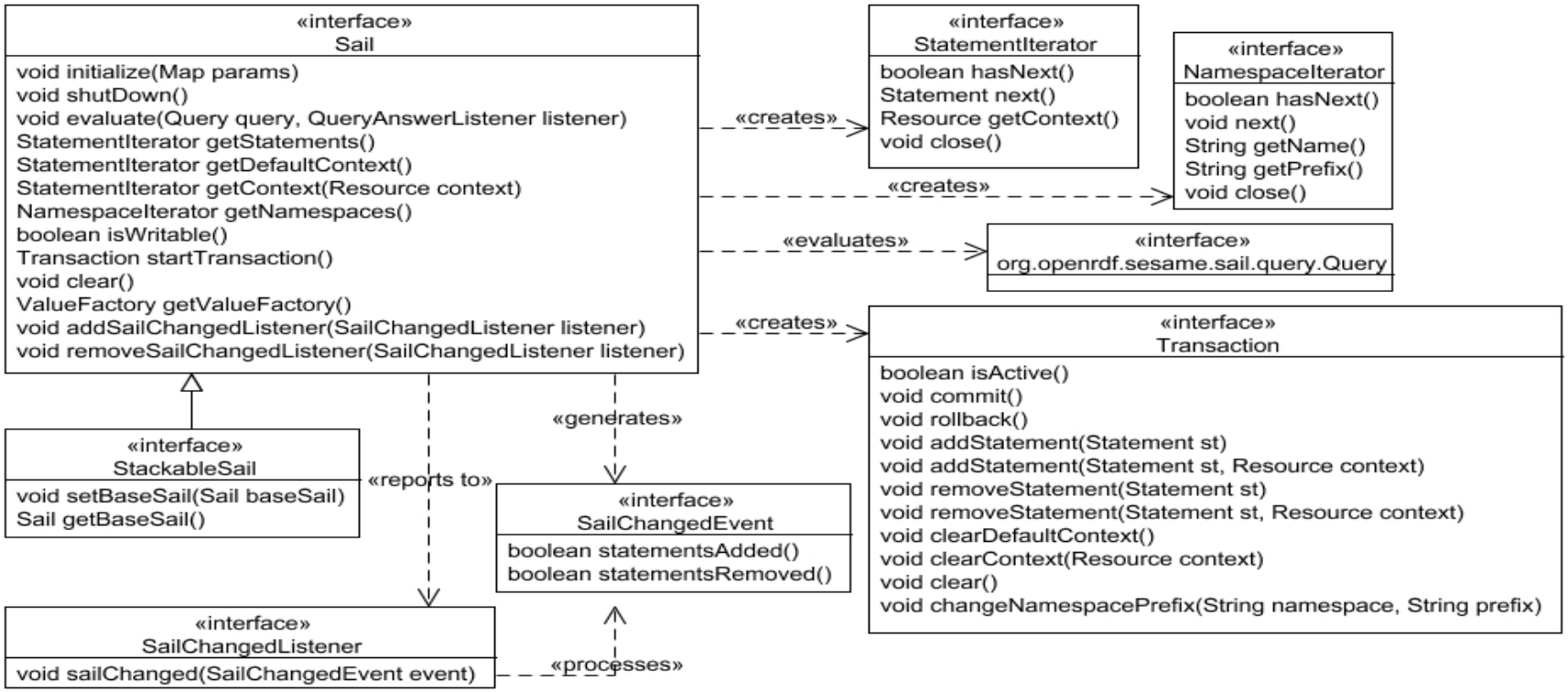
Sesame Repository Overview

- Sesame Repository API provides read/write/add/remove/query interfaces and methods to underlying SAIL
- Extensible API allows implementing vendor-specific data store code (query optimization, index creation and updates)



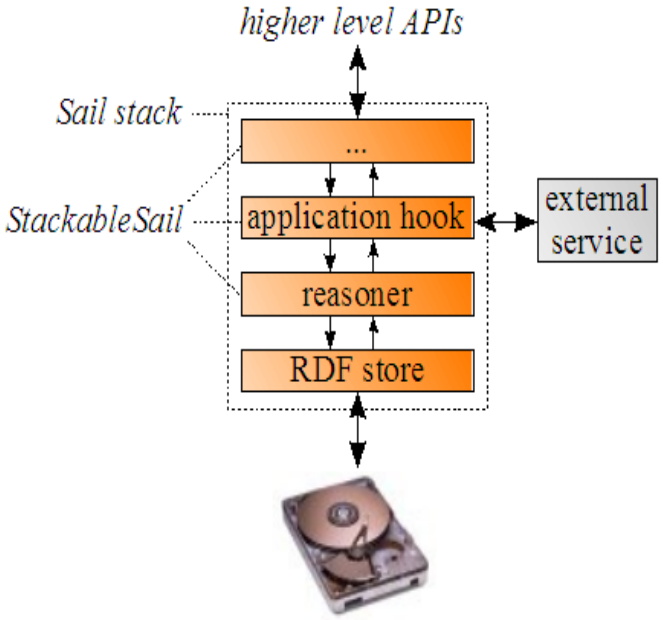
Sesame SAIL API

- Sesame SAIL API provides low-level access to data persistence storage
- Extensible API allows implementing vendor-specific data store code (result set iterators, initialization, transactions, triple inserts/deletes, etc.)



Sesame SAIL Overview

- SAIL (Storage and Inference Layer) is an abstraction layer from underlying RDF storage and processing mechanisms
- A SAIL can
  - abstract data store
  - abstract inferencing layers
  - be extended to provide any kind of specialized processing (concurrency, free text indexing, preprocessing hooks, etc.) of RDF
- SAILs can stack on top of each other to produce a variety of flexible storage, inferencing & processing architectures without breaking higher-level procedural code



Example SAIL Stack

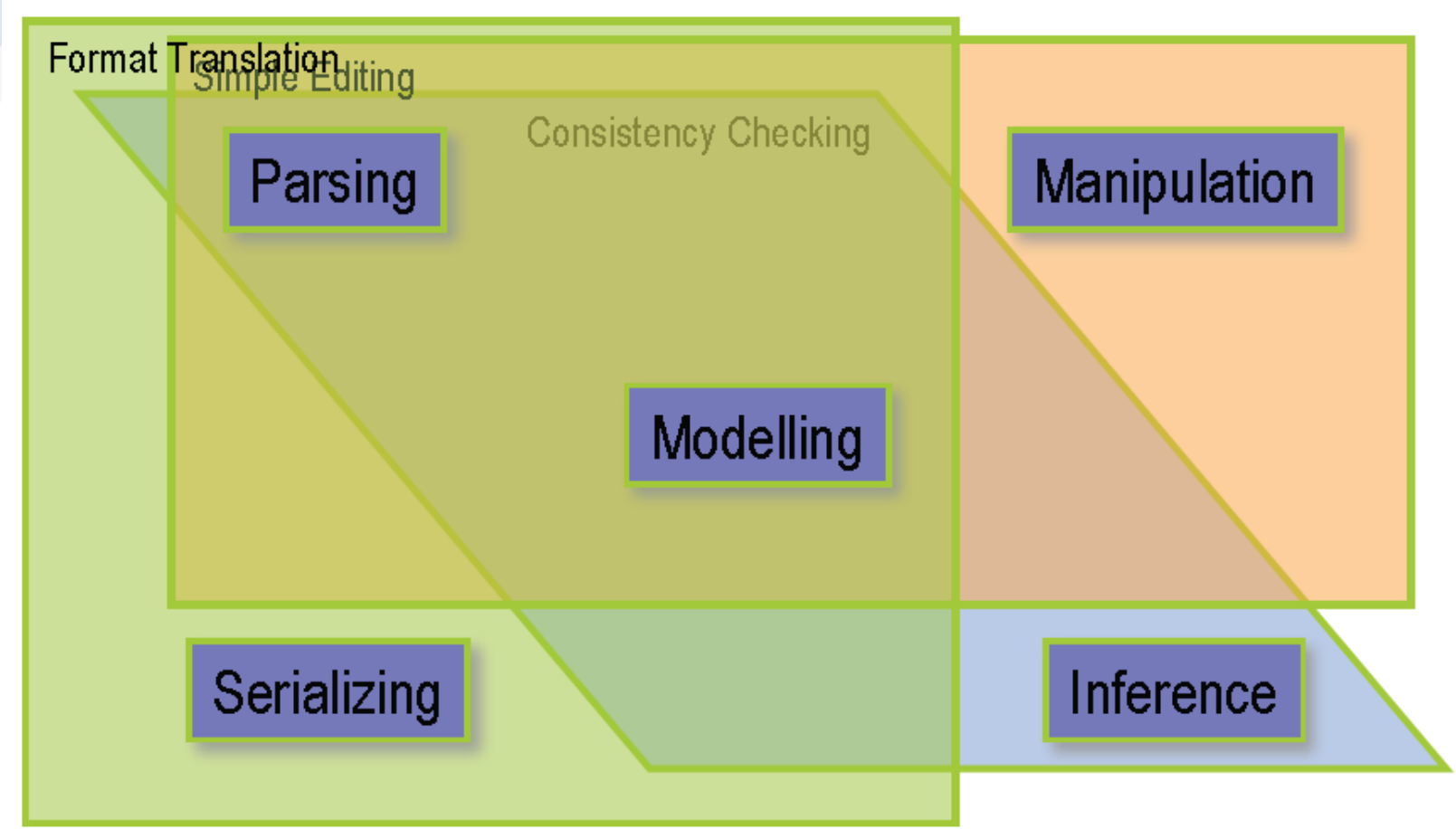
orbis TECHNOLOGIES, INC.

OWL API Overview

- Java interface and implementation for OWL
- Open source and is available under the LGPL License
  - Managed by University of Manchester
- Includes:
  - An API for OWL 2 and an efficient in-memory reference implementation
  - RDF/XML parser and writer
  - OWL/XML parser and writer
  - OWL Functional Syntax parser and writer
  - Turtle parser and writer
  - KRSS parser
  - OBO Flat file format parser
  - Support for integration with reasoners such as Pellet and FaCT++
  - Support for black-box debugging

OWL API Overview

- OWL API provides a full OWL implementation targeted primarily at OWL-DL and OWL 2



OWL API: Load an Ontology

```
// A simple example of how to load and save an ontology
// We first need to obtain a copy of an OWLOntologyManager, which, as the
// name suggests, manages a set of ontologies. An ontology is unique within
// an ontology manager. To load multiple copies of an ontology, multiple managers
// would have to be used.
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
// We load an ontology from a physical URI - in this case we'll load the pizza ontology.
URI physicalURI = URI.create("http://www.co-ode.org/ontologies/pizza/2007/02/12/pizza.owl");
// Now ask the manager to load the ontology
OWLOntology ontology = manager.loadOntologyFromPhysicalURI(physicalURI);
// Print out all of the classes which are referenced in the ontology
for(OWLClass cls : ontology.getReferencedClasses()) {
        System.out.println(cls);
}
// Now save a copy to another location in OWL/XML format (i.e. disregard the
// format that the ontology was loaded in).
// (To save the file on windows use a URL such as "file:/C:\\windows\\temp\\MyOnt.owl")
URI physicalURI2 = URI.create("file:/tmp/MyOnt2.owl");
manager.saveOntology(ontology, new OWLXMLOntologyFormat(), physicalURI2);
// Remove the ontology from the manager
manager.removeOntology(ontology.getURI());
}
```

OWL API: Add Axioms

```
OWLOntology ontology = manager.createOntology(ontologyURI);
// Now we want to specify that A is a subclass of B. To do this, we add a subclass
// axiom. A subclass axiom is simply an object that specifies that one class is a
// subclass of another class.  We need a data factory to create various object from.
// Each ontology has a reference to a data factory that we can use.
OWLDataFactory factory = manager.getOWLDataFactory();
// Get references to class A and class B. Note that the ontology does not contain ClassA
// or ClassB, we simply get references to objects from a data factory that represent
// class A and class B
OWLClass clsA = factory.getOWLClass(URI.create(ontologyURI + "#A"));
OWLClass clsB = factory.getOWLClass(URI.create(ontologyURI + "#B"));
// Now create the axiom
OWLAxiom axiom = factory.getOWLSubClassAxiom(clsA, clsB);
// We now add the axiom to the ontology, so that the ontology states that
// A is a subclass of B. To do this we create an AddAxiom change object.
AddAxiom addAxiom = new AddAxiom(ontology, axiom);
// We now use the manager to apply the change
manager.applyChange(addAxiom);
```

OWL API: Add Individuals and Properties

```
OWLDataFactory dataFactory = man.getOWLDataFactory();

// In this case, we would like to state that matthew has a father who is peter.
// We need a subject and object - matthew is the subject and peter is the
// object. We use the data factory to obtain references to these individuals
OWLIndividual matthew = dataFactory.getOWLIndividual(URI.create(base + "#matthew"));
OWLIndividual peter = dataFactory.getOWLIndividual(URI.create(base + "#peter"));
// We want to link the subject and object with the hasFather property, so use the data factory
// to obtain a reference to this object property.
OWLObjectProperty hasFather = dataFactory.getOWLObjectProperty(URI.create(base + "#hasFather"));
// Now create the actual assertion (triple), as an object property assertion axiom
// matthew --> hasFather --> peter
OWLObjectPropertyAssertionAxiom assertion =
      dataFactory.getOWLObjectPropertyAssertionAxiom(matthew, hasFather, peter);
// Finally, add the axiom to our ontology and save
AddAxiom addAxiomChange = new AddAxiom(ont, assertion);
man.applyChange(addAxiomChange);

man.saveOntology(ont, URI.create("file:/tmp/example.owl")
```

OWL API Overview: Reasoner

```
// Create a reasoner factory. In this case, we will use pellet, but we could also
// use FaCT++ using the FaCTPlusPlusReasonerFactory.
// Pellet requires the Pellet libraries (pellet.jar, aterm-java-x.x.jar) and the
// XSD libraries that are bundled with pellet: xsdlib.jar and relaxngDatatype.jar
// make sure these jars are on the classpath
OWLReasonerFactory reasonerFactory = new PelletReasonerFactory();

OWLReasoner reasoner = reasonerFactory.createReasoner(manager);

// We now need to load some ontologies into the reasoner. This is typically the
// imports closure of an ontology that we're interested in. In this case, we want
// the imports closure of the pizza ontology. Note that no assumptions are made
// about the dependency of one ontology on another ontology. This means that if
// we loaded just the pizza ontology (using a singleton set) then any imported ontologies
// would not automatically be loaded.
// Obtain and load the imports closure of the pizza ontology
Set<OWLOntology> importsClosure = manager.getImportsClosure(ont);
reasoner.loadOntologies(importsClosure);
reasoner.classify();
// We can examine the expressivity of our ontology (some reasoners do not support
// the full expressivity of OWL)
DLExpressivityChecker checker = new DLExpressivityChecker(importsClosure);
System.out.println("Expressivity: " + checker.getDescriptionLogicName());
```

orbis TECHNOLOGIES, INC.

Summary

- Semantic Web technologies represent huge potential impact to IT
- There are many similarities in Java and SemWeb technologies
- Three of the most-used APIs in the SemWeb community are Java
  - It is easy to get started!