The Need

“Most of the Web’s content today is designed for humans to read, not for computer programs to manipulate meaningfully.”

Semantic Processing

- Our goal: to be able to pose complex search tasks that utilize the semantics of pieces of information, e.g.,

I want to purchase a DVD of “Dore the Explorer” at a price lower than $10. Is such a CD available at amazon.com?

Current search agents are not suitable for such task
In order to get the feeling, try to read and understand a page written in a foreign language.

Can’t We Just Use XML?

This is what a web-page in a natural language looks like for a machine.
XML Helps

XML allows “meaningful tags” to be added to parts of the text

<name>
<education>
<work>
<private>

But to your machine, the tags look like this…. 
Schemas Help

Schemas help...

...by relating common terms between documents

But Different People use Different Schemas

Someone else has one like this....
The “semantics” is Missing

...which don’t fit in

Current Solution

Use “intelligent” agents

The Semantic-Web Approach

Content is *machine-understandable* by being bound to some *formal description* of itself (i.e. metadata)
So what is the Semantic Web?

- Humans can easily “connect the dots” when browsing the Web...
  - you disregard advertisements
  - you “know” (from the context) that one phone number in my homepage is my office phone and the other one is my fax number
  - etc.
- ... but machines can’t!
- The goal is to create a *Web of Data* that machines can exploit

Some Needs

- Better search
- Semantics in Web Services
- Data integration
Need to Improve Web Search

Spring 2013
Need for Semantics in Web Services

- If the services are ubiquitous, searching issues come up, for example: “find me the most intuitive video processing application”
- But what does it mean to be
  - “intuitive”?
  - “most intuitive”?
- People ask these questions all the time...
  - It is necessary to characterize the service not only in terms of input and output parameters...
  - ...but also in terms of its semantics

Need for Data Integration

- Databases are very different in structure & in content
- Lots of applications require managing several databases
  - after company mergers
  - combination of administrative data for e-Government
  - biochemical, genetic, pharmaceutical research
  - etc.
- Most of these data are accessible on the Web (though not necessarily public yet)
Example: Data Integration in Life Sciences

And the Problem is Real
KR Assists the Merging

SW languages add mappings and structure

Goals

- **Web of data** – provides a common data representation framework to **facilitate integrating** multiple sources to draw new conclusions
- **Increase the utility** of information by connecting it to its definitions and to its context
- More efficient information **access and analysis**
What Is Needed (Technically)?

☐ To make data machine process-able, we need:
  ▪ unambiguous names for resources that may also bind data to real world objects: URI-s
  ▪ a common data model to access, connect, describe the resources: RDF
  ▪ access to that data: SPARQL
  ▪ common vocabularies: RDFS, OWL, SKOS
  ▪ reasoning logics: OWL, Rules

☐ The “Semantic Web” is an infrastructure for interchanging and integrating data on the Web

☐ It extends the current Web (it does not replace it)

Required Applications

- Agents that search the Web and retrieve valuable information to the end user
- Web services that publish their information
- Programs that try to integrate data of different web services and to produce new results or draw new conclusions from the integrated data
Ontologies & Inference Engines

“For the semantic web to function, computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning.”


The Four Building Blocks

1. XML
2. RDF
3. Ontologies
4. Agents
XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean.

RDF – Resource Description Framework

- Meaning encoded in sets of ‘triples’: entities have properties which have values
- Entities, properties and values all have distinct URIs

“imagine that we have access to a variety of databases with information about people, including their addresses. If we want to find people living in a specific zip code, we need to know which fields in each database represent names and which represent zip codes. RDF can specify that "(field 5 in database A) (is a field of type) (zip code)," using URIs rather than phrases for each term.”

Ontologies

- Database A and Database B may use different fields to represent ‘zip code’
- Ontologies sort this out
- Ontology = ‘a document or file that formally defines the relations among terms’
- Ontologies for the web normally have
  - A taxonomy
  - A set of inference rules

Agents

“Agent based computing appears to be the appropriate paradigm to work in a complex world with multiple ontologies, fragments and multiple inferencing engines.”

The Power of Agents - Integration

“The real power of the Semantic Web will be realized when people create many programs that collect Web content from diverse sources, process the information and exchange the results with other programs. The effectiveness of such software agents will increase exponentially as more machine-readable Web content and automated services (including other agents) become available.”


Doctor’s Appointment

‘Ambient Intelligence’

“In the next step, the Semantic Web will break out of the virtual realm and extend into our physical world. URIs can point to anything, including physical entities, which means we can use the RDF language to describe devices such as cell phones and TVs.”

What is RDF?

- A part of the semantic-web activity
- RDF is a general-purpose language for representing information on the web
  - Specifically, objects and relationships
- Designed to allow computer applications to process data based on its semantics
  - Rather than displaying data to humans
- An RDF document is actually a labeled graph that is represented in XML
  - The specific language is called RDF/XML
  - W3C recommendation (Feb. 2004)
Resource Description Framework (RDF)

The data model of the Semantic Web

A schema-less data model that features unambiguous identifiers and named relations between pairs of resources

A labeled, directed graph of relations between resources and literal values

RDF Data Consists of Triplets

- RDF data is a set of statements
- Each statement is a triplet (Resource, Property, Value)
- Sometimes we refer to a triplet using the terminology of (Subject, Predicate, Object)

The author of http://www.cs.technion.ac.il/kanza/myPage.html is Yaron Kanza

Resource (Subject): myPage.html
Property (Predicate): author
Value (Object): Yaron Kanza
**RDF Data**

The basic element: **Triple (labeled edge)**

![RDF Diagram](image)

**URI-s Play a Fundamental Role**

- Anybody can create (meta)data on any resource on the Web
  - e.g., the *same* Person could be annotated through other terms
  - semantics is added to existing Web resources via URI-s
- Data exist on the Web, because it is accessible through standard Web means
- URI-s ground RDF into the Web
  - information can be retrieved using existing tools
  - this makes the “Semantic Web”, well… “Semantic Web”
Example: RDF Triples

```
<rdf:Description rdf:about="http://.../membership.svg#FullSlide">
  <axsvg:graphicsType>Chart</axsvg:graphicsType>
  <axsvg:labelledBy rdf:resource="http://...#BottomLegend"/>
  <axsvg:chartType>Line</axsvg:chartType>
</rdf:Description>
```

URI-s: Merging (data integration)

- It becomes easy to *merge* data
- Merge can be done because statements refer to the *same* URI-s
  - nodes with identical URI-s are considered identical
- Merging is a *very* powerful feature of RDF
  - metadata may be defined by several (independent) parties...
  - ...and combined by an application
  - one of the areas where RDF is *much* handier than pure XML
Data Integration Example

The XML Syntax of RDF

```xml
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
    xmlns:dc="http://purl.org/metadata/dublin_core#">
    <rdf:Description about="page.html">
        <dc:Creator>John Smith</dc:Creator>
        <dc:Title>John's Home Page</dc:Title>
    </rdf:Description>
</rdf:RDF>
```
Structured Values

page.html

dc:Title

John’s Home Page

dc:Creator

Name

Email

John Smith

js@corp.com

Containers

- Groups of things: <bag> <seq> <alt>
- <bag>: unordered list; duplicates allowed
- <seq>: ordered list; duplicates allowed
- <alt>: list of alternatives; one will be selected
DC – Dublin Core

- The mission: To make it easier to find resources using the Internet through the following activities
  - Developing metadata standards for discovery across domains
  - Defining frameworks for the interoperation of metadata sets
  - Facilitating the development of community- or discipline-specific metadata sets that are consistent with the above items

DC - Core Metadata Element Set

- Title (dc:title)
- Subject (dc:subject)
- Description (dc:description)
- Creator
- Publisher (dc:publisher)
- Contributor
- Date
- Type
- Format
- Identifier (dc:identifier)
- Source
- Language
- Relation (dcterms:isPartOf)
- Coverage
- Rights
Dublin Core

- A set of fifteen basic properties for describing generalized Web resources
  - “Title”: the name given to the resource
  - “Creator”: the person or organization primarily responsible for the resource
  - “Subject”: what the resource is about
  - “Description”: a description of the content
  - “Publisher”: the person or organization responsible for making the resource available
  - “Contributor”: someone who has provided content to the resource other than the creator
  - “Date”: date of creation or publication

Dublin Core

- “Type”: type of resource, such as home page, technical report, novel, photograph...
- “Format”: data format of the resource
- “Identifier”: URL, ISBN number, ...
- “Source”: another resource that this resource is derived from
- “Language”: the language of the content
- “Relation”: another resource and its relationship to this one
- “Coverage”: the portion of time or space described by this resource (atlases, histories, etc.)
- “Rights”: the intellectual property rights adhering to this resource, or a pointer to them
Creating RDF documents

- Manually from HTML or “user domain XML”
- With special assisting tools – like Protégé, Reggie, DC-dot, RDF for XML
- Ideally – with some automated procedure from HTML/XML documents
- Can we use XSLT for that?


**Comparison**

<table>
<thead>
<tr>
<th>Model</th>
<th>Example Format</th>
<th>Data</th>
<th>Metadata</th>
<th>Identifier</th>
<th>Query Syntax</th>
<th>Semantics (Meaning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Serialization</td>
<td>.NET CLR Object Serialization</td>
<td>Object Property Values</td>
<td>Object Property Names</td>
<td>e.g. Filename</td>
<td>LINQ</td>
<td>N/A</td>
</tr>
<tr>
<td>Relational</td>
<td>MS SQL, Oracle, MySQL</td>
<td>Table Cell Values</td>
<td>Table Column Definitions</td>
<td>Primary Key (Data Column) Value</td>
<td>SQL</td>
<td>N/A</td>
</tr>
<tr>
<td>Hierarchical</td>
<td>XML</td>
<td>Tag/Attribute Values</td>
<td>XSD/DTD</td>
<td>e.g. Unique Attribute Key Value</td>
<td>XPath</td>
<td>N/A</td>
</tr>
<tr>
<td>Graph</td>
<td>RDF/XML, Turtle</td>
<td>RDF</td>
<td>RDFS/OWL</td>
<td>URI</td>
<td>SPARQL</td>
<td>Yes, using RDFS and OWL</td>
</tr>
</tbody>
</table>

Spring 2013 236369 54
RDF Schema

RDF Schema (RDFS) enriches the data model of RDF, adding **vocabulary** and **associated semantics** for

- Classes and subclasses
- Properties and sub-properties
- Typing of properties
- Support for describing simple ontologies
- Adds an object-oriented flavor
- But with a logic-oriented approach and using “open world” semantics
RDF Schema

- Not an XML Schema!
- A “companion” specification for RDF spec
- Class, Type, subClassOf,
- Properties: domain, range
- Misc: label, comment, isDefinedBy, etc.

Classes, Resources, ...

- Basically, traditional ontologies:
  - use the term “mammal”
  - “every dolphin is a mammal”
  - “Flipper is a dolphin”
  - etc.
- RDFS defines resources and classes:
  - everything in RDF is a “resource”
  - “classes” are also resources, but...
  - they are also a collection of possible resources (i.e., “individuals”)
  - “mammal”, “dolphin”, ...
- Relationships are defined among classes/resources:
  - “typing”: an individual belongs to a specific class (“Flipper is a dolphin”)
  - “subclassing”: instance of one is also the instance of the other (“every dolphin is a mammal”)
- **RDFS formalizes these notions in RDF**
RDFS defines `rdfs:Resource`, `rdfs:Class` as nodes; `rdf:type`, `rdfs:subClassOf` as properties

- A resource may belong to several classes
  - `rdf:type` is just a property...
  - “Flipper is a mammal, but Flipper is also a TV star…”
  - i.e., it is not like a data type in this sense!
- The type information may be very important for applications
  - e.g., it may be used for a categorization of possible nodes
Inferred Properties

(#Flipper rdf:type #Mammal)
☐ is not in the original RDF data...
☐ ...but can be inferred from the RDFS rules
☐ Better RDF environments will return that triplet, too

Inference: Formal

- The RDF Semantics document has a list of (44) entailment rules:
  - “if such and such triplets are in the graph, add this and this triplet”
  - do that recursively until the graph does not change
  - this can be done in polynomial time in specific cases

- Whether those extra triplets are physically added to the graph, or deduced when needed is an implementation issue
Properties (Predicates)

- Property is a special class (rdf:Property)
- Properties are constrained by their range and domain
- Properties are also resources... (have URI’s)
- For example, (P rdfs:range C) means:
  1. P is a property
  2. C is a class instance
  3. when using P, the “object” must be an individual in C

* this is an RDF statement with subject P, object C, and property rdfs:range

Property Specification Example
Literals

- Literals may have a data type (floats, ints, etc) defined in XML Schemas, including full XML Fragments
- (Natural) language can also be specified (via xml:lang)

Example

```xml
<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base= "http://www.animals.fake/animals#">
  <rdf:Description rdf:ID="animal">
    <rdf:type
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>

  <rdf:Description rdf:ID="horse">
    <rdf:type
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#animal"/>
  </rdf:Description>
</rdf:RDF>
```

Horse is defined as subclass of animal
Example

```xml
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.animals.fake/animals#">
    <rdfs:Class rdf:ID="horse">
        <rdfs:subClassOf rdf:resource="#animal"/>
    </rdfs:Class>
</rdf:RDF>
```

Abbreviated version. Works because an RDFS class is an RDF resource.
Use `rdfs:Class` instead of `rdfs:Description` and drop the `rdf:type` information.

RDF and RDF Schema

```
<rdfs:Property rdf:ID="name">
    <rdfs:domain rdf:resource="Person"/>
</rdfs:Property>

<rdfs:Class rdf:ID="Chair">
    <rdfs:subclassOf rdf:resource="http://schema.org/gen#Person"/>
</rdfs:Class>
```

```
<g:Person rdf:ID="john">
    <g:name>John Smith</g:name>
</g:Person>
```

```
<u:Chair rdf:ID="john">
    <g:name>John Smith</g:name>
</u:Chair>
```
SPARQL Protocol and RDF Query Language

SPARQL = Query Language + Protocol + XML Results Format

- Access and query RDF graphs
- Product of the RDF Data Access Working Group
SPARQL Query

PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE {
  ?doc dc:title "SPARQL at speed" .
  ?docOther dc:creator ?c .
  ?docOther dc:title ?title
}

• Over a database of abstracts/papers:
  “Find other papers by the authors of a given paper.”

SPARQL Query

PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX shop: <http://example/shop#>
SELECT ?title
WHERE {
  FILTER regex(?title, "SPARQL") .
  ?c foaf:name ?name .
  OPTIONAL {
    ?doc shop:price ?price
  }
}

• “Find books with ‘SPARQL’ in the title. Get the authors’ name and the price (if available).”
• Multiple vocabularies
Inference

- An RDF graph may be backed by inference
  - OWL, RDFS, application, rules

```~
x rdf:type :C .
:C rdfs:subClassOf :D .
```

```sql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?type
WHERE
{
  ?x rdf:type ?type .
}
```

<table>
<thead>
<tr>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>:C</td>
</tr>
<tr>
<td>:D</td>
</tr>
</tbody>
</table>

Another Example

```~
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ldap: <http://ldap.hp.com/people#>
PREFIX foaf:
```

```sql
SELECT ?name ?email
{
  FILTER regex(?title, "SPARQL") .
  ?researcher ldap:name ?name
}
```

- “Find the name and email addresses of authors of a paper about SPARQL”
Other SPARQL Features

- Limit the number of returned results; remove duplicates, sort them, ...
- Return the full *subgraph* (instead of a list of bound variables)
- Construct a graph combining a *separate* pattern and the query results
- Use datatypes and/or language tags when matching a pattern

**OWL**
RDF Schema is Limited

- We cannot express facts such as
  - Two classes are disjoint
  - Build a class that is the union of two classes
  - Cardinality restriction
  - Scope of properties
  - Provide relationships between properties, such as transitive, unique, inverse

Ontologies (OWL)

- RDFS is useful, but does not solve all the issues
- Can a program reason about some terms? E.g.: “if «A» is left of «B» and «B» is left of «C», is «A» left of «C»?”
- Programs should be able to deduce such statements
- If somebody else defines a set of terms: are they the same?
A philosophical discipline
A branch of philosophy that deals with the nature and the organisation of reality
Science of Being (Aristotle, Metaphysics, IV, 1)
Tries to answer the questions:
  - What characterizes being?
  - Eventually, what is being?

How to Speak Ontology?

- We need a Web Ontologies Language to define:
  - more on the terminology used in a specific context
  - more constraints on properties, logical characterization of properties
  - etc.
- W3C’s Ontology Language (OWL) - A layer on top of RDFS with additional possibilities
- “OWL is now the most used KR language in the history of AI...” (Jim Hendler)
OWL

- A Web ontology language that is more expressive than RDF and RDF Schema
- Written in XML on top of RDF
- Using OWL we want to provide exact descriptions of items and the relationships between them

Why “OWL” and not “WOL”?

- Some urban legends...
  - e.g., reference to Owl from Winie the Pooh, who misspelled his name as “WOL”
  - A reference to an AI project at MIT of the mid 70’s by Bill Martin, called “One World Language”...
    - an early attempt for a KR language and associated ontology, intended to be a universal language for encoding meaning for computers
Property Restrictions in OWL

- Restriction may be by: value constraints (i.e., further restrictions on the range)
  - all values must be from a class
  - at least one value must be from a class
- cardinality constraints
  - (i.e., how many times the property can be used on an instance?)
    - minimum cardinality
    - maximum cardinality
    - exact cardinality

Property Restriction Example

“A dolphin is a mammal living in the sea or in the Amazonas”
OWL (cont.)

- **Property Characterization**
  - In OWL, one can characterize the behavior of properties (symmetric, transitive, ...)

- **Term Equivalence/Relations**
  - For classes (owl:equivalentClass, owl:disjointWith)
  - For properties (owl:equivalentProperty, owl:inverseOf)
  - For individuals (owl:sameAs, owl:differentFrom)

- **Special class owl:Ontology with special properties:**
  - owl:imports, owl:versionInfo, owl:priorVersion
  - owl:backwardCompatibleWith, owl:incompatibleWith
  - rdfs:label, rdfs:comment can also be used

---

OWL and Logic

- **OWL expresses a small subset of First Order Logic**
  - it has a “structure” (class hierarchies, properties, datatypes...), and “axioms” can be stated within that structure only.

- **Inference based on OWL is within this framework only**
Ontology and Logic

- Reasoning over ontologies
- Inferencing capabilities

\[ X \text{ is author of } Y \Rightarrow Y \text{ is written by } X \]

\[ X \text{ is supplier to } Y; \ Y \text{ is supplier to } Z \Rightarrow \]
\[ X \text{ and } Z \text{ are part of the same supply chain} \]

Cars are a kind of vehicle;
Vehicles have 2 or more wheels \( \Rightarrow \)
Cars have 2 or more wheels

However: Ontologies are Hard!

- A full ontology-based application is
  - Complex system
  - Hard to implement
  - Heavy to run
  - And not all application may need it

- Three layers of OWL are defined:
  - Lite,
  - DL (Description Logic)
  - Full
    - Decreasing level of complexity and expressiveness
FOAF: a Case Study

The *Friend of a Friend* (FOAF) project is about creating a Web of machine-readable homepages describing people, the links between them and the things they create and do.

Distributed RDF/XML records describing people, who they know, projects they work on...

FOAF - motivations

- Augment e-mail filtering by prioritizing mails from trusted colleagues
- Locate people with interests similar to yours
- 'Find an expert' in knowledge communities
- Social network analysis
- Photo co-depiction
OWL Example:
**FOAF – The way to describe yourself**
- Stands for "Friend Of A Friend"
- Provides structured links
- Information distributed & extensible
  - Name (foaf:name)
  - E-mail (Foaf:mbox)
  - Representing picture (Foaf:img)
  - Your publications (Foaf:publications)
  - Your online account (Foaf:holdsAccount)

**A simple foaf model**

```
foaf:Person
    rdf:type foaf:Person
    foaf:name John Smith
    foaf:mbox mailto:jsmith@example.com
```
JENA

- Jena is a Java framework for building Semantic Web applications
- Provides a programmatic environment for RDF, RDFS and OWL, including a rule-based inference engine
- Open source
- The Jena Framework includes:
  - An RDF API
  - Reading and writing RDF in RDF/XML, N3 and N-Triples
  - An OWL API
  - In-memory and persistent storage
  - RDQL – a query language for RDF
Jena

- A Java API for RDF
- Developed by Brian McBride of HP
- Derived from SiRPAC API
- Can parse, create, and search RDF models
- Easy to use

Install and Run Jena

- Get package from http://www.hpl.hp.co.uk/people/bwm/rdf/jena/download.htm
- Unzip it
- Setup environments (CLASSPATH)
- Online documentation
  - http://www.hpl.hp.co.uk/people/bwm/rdf/jena/javadoc/index.html
Jena package

- **jena.model**
  - Key package for application developer. It contains interfaces for model, resource, ...

- **jena.mem**
  - Contains an implementation of Jena API which stores all model state in main memory

- **jena.common**
  - Contains implementation classes

---

Jena API Structure

![Jena API Structure Diagram]
Jena interfaces

- **Model**: a set of statements
- **Statement**: a triple of \{R, P, O\}
- **Resource**: subject, URI
- **Property**: “item” of resource
- **Object**: may be a resource or a literal
- **Literal**: non-nested “object”
- **Container**: special resource, collection of things

Jena interfaces (cont.)
Ex. 1 Create Resource

static String tutorialURI = "http://hostname/rdf/tutorial/";
static String author = "Brian McBride";
static String title = "An Introduction to RDF and the Jena API";
static String date = "23/01/2001";

Model model = new ModelMem();
Resource tutorial = model.createResource(tutorialURI);
tutorial.addProperty(DC.creator, author);
tutorial.addProperty(DC.title, title);
tutorial.addProperty(DC.date, date);

Another Example

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

String ns = new String("http://www.example.com/example#");

// Create two Resources
Resource john = model.createResource(ns + "John");
Resource jane = model.createResource(ns + "Jane");

// Create the 'hasBrother' Property declaration
Property hasBrother = model.createProperty(ns, "hasBrother");

// Associate jane to john through 'hasBrother'
jane.addProperty(hasBrother, john);

// Create the 'hasSister' Property declaration
Property hasSister = model.createProperty(ns, "hasSister");

// Associate john and jane through 'hasSister' with a Statement
Statement sisterStmt = model.createStatement(john, hasSister, jane);
model.add(sisterStmt);
Ex. 2 Go through Statements

StmtIterator iter = model.listStatements();
while (iter.hasNext()) {
    Statement stmt = iter.next();
    Resource subject = stmt.getSubject();
    Property predicate = stmt.getPredicate();
    RDFNode object = stmt.getObject();
    System.out.print("(" + predicate.toString() + ", " + subject.toString() + ", " + object.toString() + ")");
}

Ex. 3 Read and Write File

Model model = new ModelMem();
String filename = "temp/tutorial4.xml";

Model.read(new FileReader(filename), "");
Model.write(new PrintWriter(System.out));
// or...

String output_filename = "temp/test.xml";
Model.write(new PrintWriter(new FileOutputStream(output_filename)));
Ex. 4 Creating/Navigating a Model

Property email = model.createProperty(tutorialURI, "emailAddress");
Resource tutorial = model.getResource(tutorialURI);
Resource author = tutorial.getProperty(DC.creator).getResource();
StmtIterator iter = author.listProperties(email);
while (iter.hasNext()) {
    System.out.println(" " + iter.next().getObject().toString());
}

Creating a Model – Another Example

// 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);
Ex. 5 Querying a Model

```c
ResIterator iter = model.listSubjectsWithProperty(DC.date, date);
while (iter.hasNext()) {
    cout << iter.next().getProperty(DC.title).getString();
}
NodeIterator iter2 = model.listObjectsOfProperty(DC.creator);
while (iter2.hasNext()) {
    cout << ((Resource)iter2.next()).getProperty(name).getString();
}
```

Containers

- Represents collections of things
  - **BAG**: unordered collection
  - **ALT**: unordered collection except first element
  - **SEQ**: ordered collection
Ex. 6 Containers

```java
Bag bag = model.createBag();
bag.add("Romeo and Juliet")
    .add("Hamlet")
    .add("Othello");
NodeIterator iter = bag.iterator();
while (iter.hasNext)
{
    System.out.println(" " + iter.next().toString());
}
model.write(new PrintWriter(System.out));
```

Datatype Properties

```java
// 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);
```
Object Properties

// Create object property 'hasSibling'
ObjectProperty hasSibling = ontModel.createObjectProperty(ns + "hasSibling");

// Domain and Range for 'hasSibling' is 'Person'
hasSibling.setDomain(person);
hasSibling.setRange(person);

// Add statement 'John hasSibling Jane'
// and 'Jane hasSibling John'
Statement siblings1 = ontModel.createStatement(john, hasSibling, jane);
Statement siblings2 = ontModel.createStatement(jane, hasSibling, john);
ontModel.add(siblings1);
ontModel.add(siblings2);

Property Restrictions

// Create object property 'hasSpouse'
ObjectProperty hasSpouse = ontModel.createObjectProperty(ns + "hasSpouse");
hasSpouse.setDomain(person);
hasSpouse.setRange(person);
Statement spouse1 = ontModel.createStatement(bob, hasSpouse, jane);
Statement spouse2 = ontModel.createStatement(jane, hasSpouse, bob);
ontModel.add(spouse1);
ontModel.add(spouse2);

// Create an AllValuesFromRestriction on hasSpouse:
// MalePersons hasSpouse only FemalePerson
AllValuesFromRestriction onlyFemalePerson = ontModel.createAllValuesFromRestriction(null, hasSpouse, femalePerson);

// A MalePerson can have at most one spouse -> MaxCardinalityRestriction
MaxCardinalityRestriction hasSpouseMaxCard = ontModel.createMaxCardinalityRestriction(null, hasSpouse, 1);

// Constrain MalePerson with the two constraints defined above
malePerson.addSuperClass(onlyFemalePerson);
malePerson.addSuperClass(hasSpouseMaxCard);
Defined Classes

// Create class 'MarriedPerson'
OntClass marriedPerson = ontModel.createClass(ns + "MarriedPerson");
MinCardinalityRestriction mincr = ontModel.createMinCardinalityRestriction(null, hasSpouse, 1);

// A MarriedPerson ⇔ A Person, AND with at least 1 spouse
// A list must be created, that will hold the Person class
// and the min cardinality restriction
RDFNode[] constraintsArray = { person, mincr }; RDFList constraints = ontModel.createList(constraintsArray);

// The two classes are combined into one intersection class
IntersectionClass ic = ontModel.createIntersectionClass(null, constraints);

// 'MarriedPerson' is declared as an equivalent of the
// intersection class defined above
marriedPerson.setEquivalentClass(ic);

Reasoning

• 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

- Objects of the `OntModelSpec` class are used to form model specifications
  - Storage scheme
  - Inference engine
  - Language profile (RDF, OWL-Lite, OWL-DL, OWL Full, DAML)
- Jena provides predefined `OntModelSpec` objects for basic Model types
  - e.g., the `OntModelSpec/owl-dl_mem` object is a specification of OWL-DL models, stored in memory, which use no reasoning.

```java
// PelletReasonerFactory is found in the Pellet API
Reasoner reasoner = PelletReasonerFactory.theInstance().create();
// Obtain standard OWL-DL spec and attach the Pellet reasoner
OntModelSpec ontModelSpec = OntModelSpec/owl-dl_mem;
ontModelSpec.setReasoner(reasoner);
// Create ontology model with reasoner support
OntModel ontModel = ModelFactory.createOntologyModel(ontModelSpec, model);
```
Reasoning

- Apart from the reference to a Reasoner object, no further actions are required to enable reasoning.
- OntModels without reasoning support will answer queries using only the asserted statements, while OntModels with reasoning support will infer additional statements, without any interaction with the programmer.

```java
// MarriedPerson has no asserted instances
// However, if an inference engine is used, two of the three
// individuals in the example presented here will be
// recognized as MarriedPersons
OntClass marriedPerson = ontModel.getOntClass(ns + "MarriedPerson");
ExtendedIterator married = marriedPerson.listInstances();
while(married.hasNext()) {
    OntResource mp = (OntResource)married.next();
    System.out.println(mp.getURI());
}
```
SPARQL query processing

- 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

SPARQL query execution example

```java
// 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 {\n" + "?married rdf:type :MarriedPerson.\n" + "?married :hasSpouse ?spouse." + "}\n";

// 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);
Query q = QueryFactory.create(queryString);

// Execute query and obtain result set
QueryExecution qexec = QueryExecutionFactory.create(q, dataset);
ResultSet resultSet = qexec.execSelect();
```
SPARQL Query Execution Example

```java
// 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());
}
```

Notes

- Jena can be used to manage existent ontologies, or create ontologies from scratch
- Reasoning with existent data in order to obtain inferred knowledge
  - Inference engines must provide implementations of a specific Java interface
  - For complex ontologies, reasoning may slow down your application, especially if data is inserted or removed regularly from the ontology
  - It is important to know when an inference engine is actually needed
Joseki - a SPARQL Server for Jena

- **Joseki** – the Jena RDF Server
  - A server for publishing RDF models on the web
  - Models have URLs and they can be accessed by HTTP GET
  - Joseki is part of the Jena RDF framework
  - Joseki is an HTTP and SOAP engine supporting the SPARQL Protocol and the SPARQL RDF Query language

**Joseki Features:**
- RDF Data from files and databases
- HTTP (GET and POST) implementation of the SPARQL protocol
- SOAP implementation of the SPARQL protocol


Sesame

- Sesame is a Java framework for storing, querying and inferencing for RDF. It can be deployed as a web server or used as a Java library. Features include several query languages (SeRQL and SPARQL), inferencing support, and RAM, disk, or RDBMS storage.

Spring 2013 236369 126

Spring 2013 236369 127
Links

• http://www.w3.org/RDF/

Jena References

• Jena Web Site
  • http://www.hpl.hp.co.uk/people/bwm/index.html

• Jena tutorial
  • http://bmcb.btinternet.co.uk/2001/rdf/jenatutorial/
  • http://www.xml.com/pub/a/2001/05/23/jena.html

• RDF Model and Syntax Specification
  • http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
SPARQL Links

- Jena: Java and .Net Semantic Web Framework
- SPARQL Query
  - [http://jena.sourceforge.net/ARQ](http://jena.sourceforge.net/ARQ)
- SPARQL Protocol
  - [http://www.joseki.org](http://www.joseki.org)
- SquirrelRDF: Access legacy SQL:
  - [http://jena.sourceforge.net/SquirrelRDF](http://jena.sourceforge.net/SquirrelRDF)