

Semantic Web Engineering

Matthias Hert
hert@ifi.uzh.ch

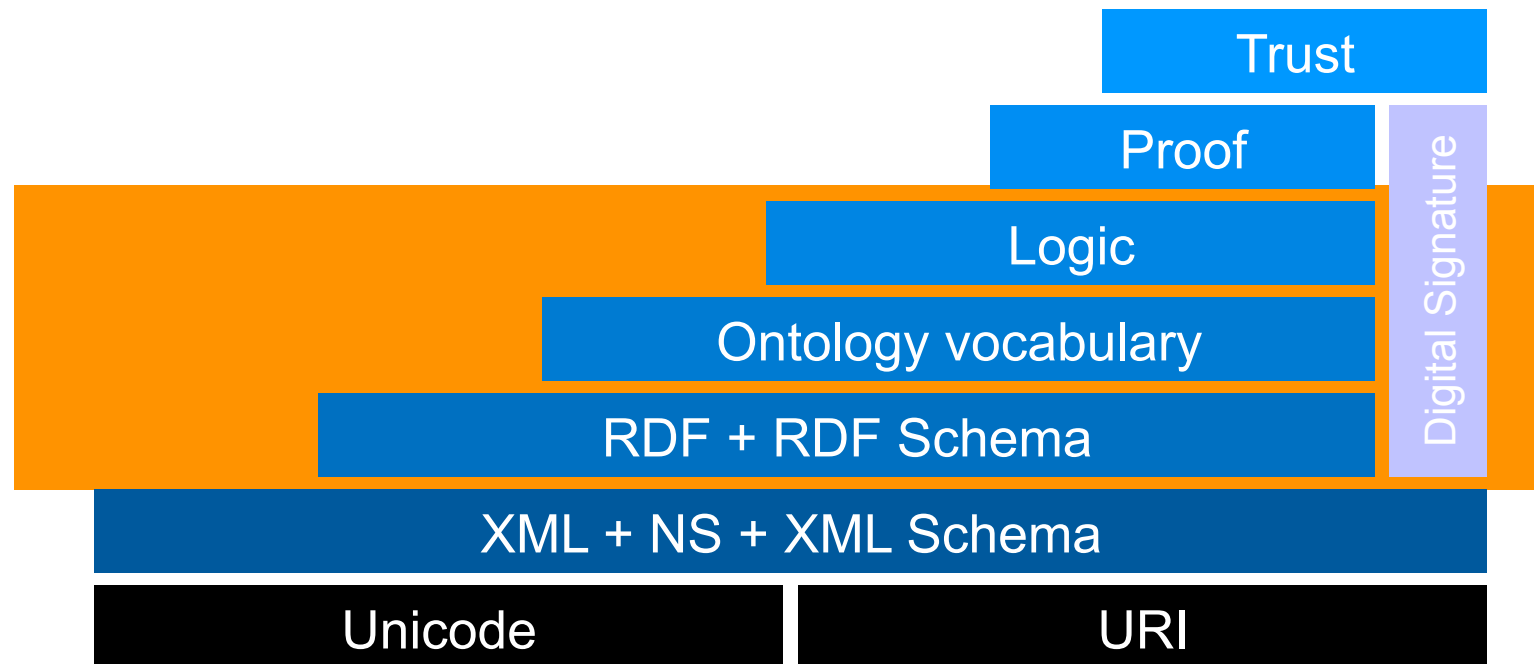
Fr. 10:15-12:00, Room 2.A.10



University of Zurich
Department of Informatics



Web Ontology Language



Programming with RDF and OWL

- Java Platform:

- **Jena**: Open source project initiated by HP
<http://jena.sourceforge.net/>
- **Sesame**: Open source framework for storage, inferencing and querying of RDF data
<http://www.openrdf.org/>
- **OWL API**: Open source API for OWL
<http://owlapi.sourceforge.net/>

Programming with RDF and OWL

- .NET Platform: **SemWeb**
 - C# based RDF library for the .NET platform
 - RDF/XML & N3 input/output, storage, querying & (limited) inferencing
<http://razor.occams.info/code/semweb/>
- PHP: **RAP**
 - RAP is a software package for parsing, querying, manipulating, serializing and serving RDF models
<http://www4.wiwiss.fu-berlin.de/bizer/rdfapi/>
- C: **Redland**
 - <http://librdf.org/>
- ...

Jena Framework

- Jena is a Java framework for building Semantic Web applications.
- It provides a programmatic environment for RDF, RDFS and OWL, including a rule-based inference engine.
- Jena is open source and grown out of work of the HP Labs Semantic Web Program.
- The Jena Framework includes:
 - An RDF API
 - Reading and writing RDF in RDF/XML, N3, N-Triples, Turtle
 - An OWL API
 - In-memory and persistent storage
 - SPARQL and RDQL – query languages for RDF

The Jena RDF API

- Statement centric methods for manipulating an RDF model as a set of RDF triples
- Resource centric methods for manipulating an RDF model as a set of resources with properties
- Cascading method calls for more convenient programming
- Built in support for RDF containers - Bag, Alt and Seq
- Enhanced resources - the application can extend the behavior of resources
- **Integrated parsers** and writers for RDF/XML, N3, N-Triples and Turtle
- Support for typed literals

Jena:

Creating Graphs and Statements

- An RDF graph in Jena is called Model

```
// create an empty Model
```

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

- Create a resource

```
Resource johnSmith = model.createResource("http://  
somewhere/JohnSmith");
```

- Create a property

```
Property hasName = model.createProperty("http://  
example.com/terms#hasName");
```

- Add the property `hasName` to the resource `johnSmith`

```
johnSmith.addProperty(hasName, "John Smith");
```



Jena: Built-in Properties

- Jena has built-in Properties for common ontology vocabularies
 - i.e. RDF, RDFS, OWL, vCard, DC, etc.
- Defined in Java package: `com.hp.hpl.jena.vocabulary`
- vCard Ontology
 - Ontology vocabulary to define electronic business cards
 - Originally defined in RFC 2426
 - Used namespace:
`http://www.w3.org/2001/vcard-rdf/3.0#`
 - vCard Ontology definition:
<http://www.w3.org/TR/vcard-rdf>
 - i.e. properties such as: `#Given`, `#Family`, `#N (Name)`, `#FN (FullName)`

Complex Graph with blank nodes

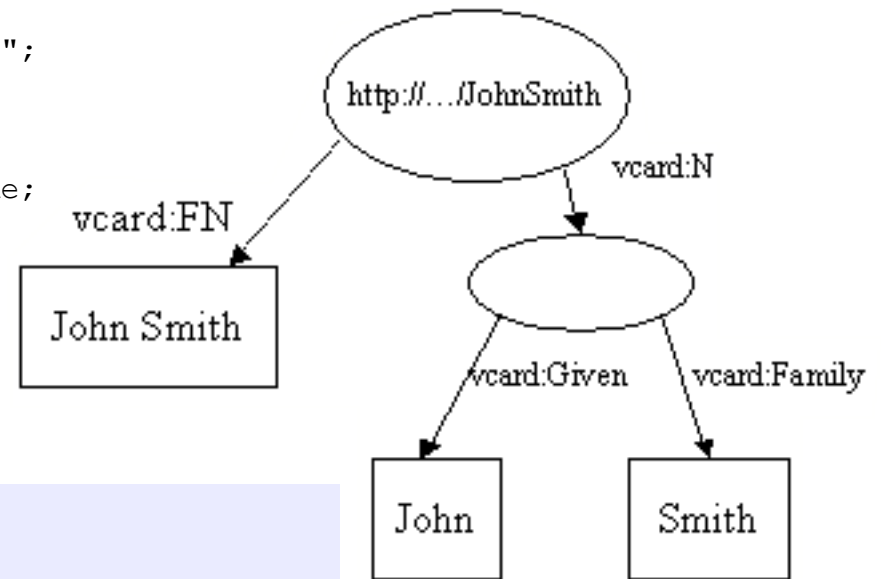
```
// some definitions
String personURI    = "http://somewhere/JohnSmith";
String givenName    = "John";
String familyName    = "Smith";
String fullName      = givenName + " " + familyName;
```

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

```
// create the resource
// and add the properties cascading style
```

```
Resource johnSmith
    = model.createResource(personURI)
```

```
Resource johnSmith
    = model.createResource(personURI)
        .addProperty(VCARD.FN, fullName)
        .addProperty(VCARD.N,
            model.createResource()
                .addProperty(VCARD.Given, givenName)
                .addProperty(VCARD.Family, familyName));
```



Jena: Statements

- Each arc in an RDF Model is called a statement.
- Each statement asserts a fact about a resource.
- A statement has three parts:
 - The **subject** is the resource from which the arc leaves
 - The **predicate** is the property that labels the arc
 - The **object** is the resource or literal pointed to by the arc
- The `Jena Model` interface defines a `listStatements()` method which returns an `StmtIterator`, a subtype of Java's `Iterator` over all the statements in a `Model`.
- `StmtIterator` has a method `nextStatement()` which returns the next statement from the iterator.
- The `Statement` interface provides accessor methods to the subject, predicate and object of a statement.

Jena: Writing and Reading RDF

- The Jena `Model` interface defines a `write()` method which writes the RDF graph to an Output Stream.
- Syntax:
 - `write(OutputStream os, String base, String RDFSyntax)`
- Syntax currently supported by Jena:
 - `"RDF/XML"` (`"RDF/XML-ABBREV"`), `"Turtle"` (`"TTL"`), `"N-TRIPLE"` and `"N3"`
- The Jena `model` interface defines also a `read()` method to read RDF from an Input Stream.
- Syntax:
 - `read(InputStream os, String base, String RDFSyntax)`
- Supported syntax like with `write()`

Jena: Navigating and Manipulating the RDF graph

- Jena provides many methods to navigate the RDF graph
 - i.e., `model.getResource(uri)`,
`stat.changeObject(some_value_or_URI)`,
`stat.getObject()`, etc.
 - For more look at the Jena API
 - <http://jena.sourceforge.net/javadoc/index.html>
- With `addProperty()` new properties can be added to a Resource.

Jena: Querying the RDF Graph

- The `Model.listStatements()` method, which lists all the statements in a model. **Its use is not recommended on very large Models.**
- `Model.listSubjects()` is similar, but returns an iterator over all resources that have properties, i.e., are the subject of some statement.
- `Model.listSubjectsWithProperty(Property p, RDFNode o)` will return an iterator over all the resources which have property `p` with value `o`.
 - Parameters can be null as a wildcard
- It is better to use a **Selector** to query RDF graphs.

The Jena Selectors Interface

- The `SimpleSelector` implements the `Selector` Interface.
- The `SimpleSelector` constructor takes three arguments

```
Selector selector =  
    new SimpleSelector(subject, predicate, object)
```
- This selector will select all statements with a subject that matches subject, a predicate that matches predicate and an object that matches object.
 - If a `null` is supplied in any of the positions, it matches anything;
 - otherwise they match corresponding equal resources or literals.

SimpleSelector Example

- Select all the resources with a `VCARD.FN` property whose value ends with "Smith"

```
StmtIterator iter = model.listStatements(  
    new SimpleSelector(null, VCARD.FN, (RDFNode) null) {  
        public boolean selects(Statement s)  
        {return s.getString().endsWith("Smith");}  
    });
```

- This sample code uses a neat Java technique of **overriding a method definition inline** when creating an instance of the class.
 - Here the `selects(...)` method checks to ensure that the full name ends with "Smith".
 - It is important to note that filtering based on the subject, predicate and object arguments takes place before the `selects(...)` method is called, so the extra test will only be applied to matching statements.

Reasoning in Jena

- Jena comes with several build in reasoners (RDFS, OWL)
- Other reasoners such as pellet can be used
- More on the Jena reasoning support:
 - <http://jena.sourceforge.net/inference/>

Simple Example: RDFS Reasoning

```
String termsNS = "http://example.com/terms#";
String instanceNS = "http://example.com/terms#";

Model rdfsExample = ModelFactory.createDefaultModel();
Property subProp = rdfsExample.createProperty(termsNS, "subProp");
Property superProp = rdfsExample.createProperty(termsNS, "superProp");
rdfsExample.add(subProp, RDFS.subPropertyOf, superProp);
rdfsExample.createResource(instanceNS + "a").addProperty(subProp, "foo");

rdfsExample.write(System.out, "Turtle");

InfModel inf = ModelFactory.createRDFSModel(rdfsExample);

Resource a = inf.getResource(instanceNS+"a");
System.out.println("Statement: " + a.getProperty(superProp));
```

Simple Example: OWL Reasoning

```
Model schema = ModelFactory.createDefaultModel();
Model data = ModelFactory.createDefaultModel();

schema.read("file:JenaReasoningExample.owl", "RDF/XML");
data.read("file:data.rdf", "Turtle");

Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
reasoner = reasoner.bindSchema(schema);
InfModel infmodel = ModelFactory.createInfModel(reasoner, data);

infmodel.write(System.out, "RDF/XML");
```