2. RDF Resource Description Framework

- RDF (Instances)
  - Statements about Resources (addressable by an URI) and literals (XML data)
  - Statements are of the form: subject property object
  - Like simple natural language sentences
  - RDF statements are themselves resources (reasoning about RDF)
  - Properties define relationships to other resources or atomic values

- RDF-Schema
  - Data model to specify schemas for RDF instances
  - Which properties are applicable for which objects with which subjects
  - Defines "grammar" and "vocabulary" for semantic domains (ontology language)

- Relation RDF vs. RDFS comparable to well-formed vs. valid XML

RDF is a standard supported by the W3C (http://www.w3.org/RDF/) to represent metadata. It is a fairly simple, graph-oriented data model to annotate any kind of XML document.

RDF consists of two parts: a language for the metadata instances (RDF), which allows to connect simple « sentences » with document parts that are addressed by Universal Resource Identifiers (URI – we will call these document parts in the following resources), of which URL’s are the most important example. And a language for specifying schemas for RDF Instances, that defines the possible vocabularies and the grammar of how these sentences may be formed. So the situation is very similar as with well-formed XML (instances) and XML-DTD (schemas).

The RDF model is in some sense similar to the entity-relationship (ER) model. Entities correspond to resources and relationships correspond to properties. The main difference is that RDF requires that relationships are directed, and carry the semantics that the resource from which the (directed) relationship emerges is assigned a property with the value to which the relationship points. This reflects the intention to use RDF to associated metadata (the value) with data (the source of the relationship).

Typical applications include PICS (annotating documents with information on the suitability of the content for certain groups, e.g. like the movie rating system) and Dublin Core (annotating documents with basic bibliographic information).

It is important to keep in mind that RDF is was only specified in a « semi-formal » manner, that means the XML encoding is well specified, however the underlying meaning of the syntax is given informally. Tools for processing (parsing, editing, storing, querying) RDF are developing at a fast pace.
RDF instances are RDF statements. We can view RDF instances in three different ways: we can view them like natural language sentences, where the subject is a URIs and the object can be either URI’s or Strings, we can view them as directed graphs, where the subject and object are considered as nodes and the predicate is considered as link, or we can view them as XML documents where the RDF statement is encoded into an XML document. For each purpose (simply understanding the meaning, processing and algorithms for manipulating the structure, or representation for exchange and storage), one of the forms is most suitable.

In the graph representation an ellipsis is used to represent resources (identified by a URI) and rectangles to represent literals (atomic XML values).
RDF Syntax

- Many syntactic varieties possible
- Basic form

For encoding RDF there exist many syntactic variations which make reading RDF documents sometimes rather confusing. Here we summarize the most important variants. The basic pattern of encoding is as follows: the subject is referenced in an element called rdf:Description. This element is the root of the document fragment representing the RDF statement. In the content of this element one finds one (or more) predicates, represented by elements, e.g. s:Creator. The content of this element in turn is the object of the statement. If the object is not a literal, one can alternatively represent the object as an attribute of the predicate element. Also, both the predicate and the object can be encoded into the element representing the statement, as it is shown for s:Date.
XML Namespaces

• An **XML namespace** is a collection of names (markup vocabulary)
  - identified by a URI reference
  - agreed element and attribute names

```xml
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
```

• **URIs** are used just as unique identifiers, nothing else
  - in particular they do not refer to a DTD or schema

• **Uses**
  - universally agreed names
  - combination of names from different DTDs without name conflicts
  - but not combination of different DTDs

XML namespaces are a convention to distinguish different vocabularies for different applications in order to avoid names collisions. Consider just of how many XML applications would use the element name "name" and what happens if an application encounters this element name in documents originating from different sources without being able to distinguish the origin of the name. Therefore element and attribute names are prefixed by a label that is unique for a namespace. The problem is of course of how to obtain these unique labels. The solution is very simple: one uses Universal Resource Identifiers (typically URLs). But one must be careful here: the only purpose of using a URI (URL) is to have a unique prefix. There is nothing else associated with the URL, in particular the corresponding URL may not exist, it does not contain any information like a DTD or a Schema (though often useful information pertaining to the namespace is found under its specific URL).

Namespaces are important in order to provide universally agreed names with unambiguous semantics that can be exploited by applications. They also allow to combine element names from different DTDs which would otherwise be in conflict which each other. But the namespace concept is not related to the problem of combining definitions from different DTDs in any way.
Declaration

- Declaration of
  - Default namespace: xmlns (all names without prefix are supposed to be from there)
  - Identification by prefix: xmlns:ns (ns is the prefix)
- Can be declared in different positions
  - In internal DTD by using default attributes
  - In document by using attributes

```xml
<?xml version= "1.0"?><!
<!-- element names without prefix belong to "books" -->
<book xmlns='urn:loc.gov:book'
     xmlns:isbn='urn:ISBN.0-395-36341-6'>
  <title>XML Handbook</title>
  <isbn:number>1591240349</isbn:number>
</book>
```

Within the scope of one element one can always declare one default namespace, to which then all then names belong that do not have a prefix. In addition, an arbitrary number of other namespaces can be distinguished by a prefix. The namespace declaration can be given both in the DTD, then it is a default attribute, or within the document by using attributes. We see in the example all the cases illustrated:

In the example document an attribute xmlns is used to declare the default namespace, such that for example the element title is from the namespace "urn:loc.gov:bok" and a second namespace isbn is declared.

In the example DTD the element type x is equipped with the default namespace http://www.jclark.com/. Note that the keyword FIXED is used in the attribute declaration in order to express that it is an unchangeable attribute value.
RDF Complex Values

• Use an intermediate resource

Single values for the objects of an statement are just the simplest possible case. In general, if we have a complex object about which we want to make a statement involving multiple properties we create a new intermediate resource that combines the multiple properties and we can make then statements about this new complex object. In the XML encoding this can be represented by directly inlining the complex object into the content of the statement.
RDF Containers

- **Containers**
  - Bag (unordered)
  - Seq (ordered)
  - Alt (alternatives)

- **Quantifiers**
  - about: Urs is author of the talk (consisting of many slides)
  - aboutEach: Urs is author of each slide of the talk

Statements can be made not only on single resources, but as well on collections of resources. For that purpose RDF provides the container concept. Containers are special resources of one of three container types that are specified in the RDF standard. A container resource is then associated with a set of other resources. By creating a statement using the container object as "object", one can express statements made about the set of objects. More precisely, one can specify whether the statement is a statement of the set of objects "as a whole" or a statement that applies to each element of the set individually (what the consequences of this distinction are is not further specified in RDF).

There exist three different types of containers: bags which are unordered multi-sets (= sets with multiple occurrences of the same resources), sequences which are ordered sets (i.e. lists) of resources and alternatives which is a single resource that is to be chosen out of a given set. The property labels can be used to impose an order of elements of the set, by using labels _1, _2 etc. If the order is irrelevant one can use the alternative syntax rdf:li instead of rdf:_1, rdf:_2 etc.
Typing Resources

- Resources can be associated with a type by using **rdf:type** (a special property)

RDF allows to categorize documents into different classes. For doing that one associates with the resource that should be categorized another resource, that represents the category, with a special property **rdf:type** which is defined as part of the RDF specification. We will see later, when introducing RDF schema, what are possible consequences of typing. With respect to encoding, the type property can either be represented as any other property, or one can use a special abbreviated syntax, where the name of the type becomes the element name of the element embracing the statement.
Creating New Resources

- New RDF resources are created by using `rdf:ID` (a special property)

In this example the `rdf:Description` element has an `rdf:ID` attribute instead of an `rdf:about` attribute. Using `rdf:ID` indicates that we are describing a new resource, identified by the value of the `rdf:ID` attribute ("12345" in this case), rather than referring to an existing resource defined somewhere else. This resources can from there be referenced later by other RDF statements.
In RDF everything is a resource. In particular, RDF statements are resources and therefore should be annotated. From the viewpoint of the Semantic Web this is in fact extremely important. Since annotations do not express absolute truth but rather interpretations of data, and interpretations may be different, it is important to foresee the capability of annotating annotations (such as illustrated in the simple example). This allows to comment on annotations, to agree or dispute them etc.

Structurally, when inspecting the graph representation of RDF, it is not immediately clear of how to treat a statement as a resource, since a statement consists of three structural elements, the subject, the object, and the predicate. But we can apply the same "trick" as we did already for complex objects and collections. We introduce an abstract object which serves as representative for the statement, and make this object a representative by properly specifying its properties. This process is called reification. As for the properties of the reified statement it is straightforward to connect it to its subject and object as they are both resources. Also the type of the object is determined through a property rdf:type pointing to the special resource rdf:statement representing the category of statements. The predicate requires the introduction of a further object representing it. As we will see later this new resource representing the predicate is in fact an object that belongs to an RDF schema. For reification RDF introduces special properties rdf:object, rdf:subject and rdf:property, which are part of the RDF specification.

By reifying a statement one creates a new anonymous resource (this can be recognized from the fact that no identifier is found in the node representing the reified statement).
RDF Reification - Syntax

- The statement has an anonymous resource as subject, namely the reified statement which is fully characterized by its properties!

```xml
<rdf:RDF>
  <rdf:Description>
    <rdf:subject resource="http://www.doc.ch"/>
    <rdf:predicate resource="http://description.org/schema#Creator"/>
    <rdf:object>Urs Giger</rdf:object>
    <rdf:type resource="http://www.w3.org/TR/WD-rdf-syntax#Statement"/>
    <dc:Creator>Irma Müller</dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

The XML encoding of reified statements follows the principles that have been introduced before. The annotation statement is represented as a complex, anonymous object.
We now introduce RDF Schema. RDF Schema provides two basic concepts.

1. the possibility to categorize RDF resources, into classes.
2. the possibility to constrain the usage of properties for classes, i.e. which classes can participate as subjects and objects in statements using a specific property.

First we introduce the classification mechanism. Classes are represented themselves as resources, which are of type rdfs:Class, a special class introduced in the RDFS specification. The type property rdf:type is used as before to indicate the type of class resources. If an application resource is of a specific application type then the resource is connected to the class resource via the rdf:type predicate. In the example two cases can be seen, the resource with ID "Karl Aberer" belongs to class (or is of type) Person and the resource "Married" is of type MaritalStatus. Between different classes a subclass relationship can be specified, by using the attribute rdfs:subClassOf. The intended semantics is that any resource belonging to the subclass also belongs to the superclass (containment relationship).

An interesting point is to see of how RDFS treats it's own modeling constructs within an RDF meta-schema that specifies the RDFS model in terms of the RDFS model itself. In RDF everything is a resource. Therefore in particular also resources that are classes are resources, and thus the rdfs:Class resource is a subClass of rdfs:Resource. Also application classes, such as "Animal", contain resources, and thus also these classes are subclass of rdfs:Resource. On the other hand, the type of a resource is indicated by connecting it by a rdf:type property to the resource that represents the type. Since rdfs:Resource is a class it is connected via the rdf:Type property to the rdfs:Class resource. This produces the cyclic structure that we can observe within the RDF meta-class schema (in fact in the figure only a small fragment of the RDFS meta-class schema is displayed).
The second important concept of RDFS is the possibility to define properties and their usage. RDF properties connect resources. As everything in RDF, RDF properties are resources themselves. Thus, they are represented in an RDF schema as resources. For example maritalStatus is a resource representing a property. In RDF schema it is now possible to constrain the usage of properties as follows: by connecting the property resource through the property rdfs:domain to a class resource, one specifies that the subject when using this property must originate from that class, i.e. be of the type of this class. Similarly for the object, the range can be constrained using rdfs:range. Ranges can also be atomic type, in that case one connects the property resource to (predefined) resources representing the data type of the atomic type, in the example above, this is STRING.

The RDFS model bears a lot of similarity with object-oriented models (or type specifications in the context of OO programming languages). However a fundamental difference is that properties (attributes in OO terminology) are defined independently of classes.
RDF Schema - Syntax

```xml
<rdf:RDF xml:lang="en"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <rdfs:Class rdf:ID="Person">
        <rdfs:subClassOf rdf:resource="http://www.w3.org/classes#Animal"/>
    </rdfs:Class>
    <rdf:Property ID="maritalStatus">
        <rdfs:range rdf:resource="#MaritalStatus"/>
        <rdfs:domain rdf:resource="#Person"/>
    </rdf:Property>
    <rdf:Property ID="name">
        <rdfs:range rdf:resource="http://www.w3.org/classes#String"/>
        <rdfs:domain rdf:resource="#Person"/>
    </rdf:Property>
    <rdf:Property ID="age">
        <rdfs:range rdf:resource="http://www.w3.org/classes#Integer"/>
        <rdfs:domain rdf:resource="#Person"/>
    </rdf:Property>
    <rdfs:Class rdf:ID="MaritalStatus"/>
    <MaritalStatus rdf:ID="Married"/>
    <MaritalStatus rdf:ID="Divorced"/>
    <MaritalStatus rdf:ID="Single"/>
    <MaritalStatus rdf:ID="Widowed"/>
</rdf:RDF>
```

Since RDF schemas are expressed as RDF statements they can be encoded into XML along the same principles that we have introduced for RDF statements earlier. This example shows the complete encoding of the RDF schema we have used in our example before. Throughout the schema the abbreviated syntax for statements is used, replacing the element name "Description" by the corresponding class name of the subject of the description. Note of how using the ID attribute in the description elements, in the RDF schema new resources are introduced. They can be referred to from other statements using the newly introduced identifier and prepending #. The specification of the class "MaritalStatus" includes the creation of the complete extension of the class, enumerating all possible values. Strictly speaking, the statements creating the instances of class are not part of the schema level but part of the instance level of RDF.
RDF Schema Inheritance

- **rdfs:subClassOf**
  - A subClassOf B: Every instance of A is also instance of B
  - transitive, not reflexive, anti-symmetric (no cycles!),
  - M:N: a class can have arbitrarily many subclasses and superclasses
  - Subclass has all properties of the superclass

- **rdfs:subPropertyOf**
  - P1 subPropertyOf P2: If A has Property P1 with value B then it has also value B with Property P2
  - Example: Irma has Property Father with value Urs, and Father subProperty AParent implies Irma has Property AParent with value Urs

Similar to object-oriented modelling, RDF provides inheritance mechanisms in order to support the reuse of specifications. RDF Schema provides both for type inheritance and value inheritance.

1. **Type inheritance** is indicated by the rdfs:subClass property. If a class A is subclass of a class B this implies two things: first, every instances of A is also an instance of B, and second, every property that is specified for A can also be used for B (or is also specified for B). Since the subclass relationship has the subset semantics it is transitive, but must not contain cycles.

2. **Property inheritance** is indicated by connecting two properties by the property rdfs:subPropertyOf. If a property P1 is subproperty of another property P2, this has the following meaning. Every resource A that has property P1 (i.e. is the subject of a statement involving that property), and this property has value B (i.e. B is the object of the statement), then it has also property P2, an property P2 has the value B.
### Comparison XML - RDF

<table>
<thead>
<tr>
<th>Aspect</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document/Instance</td>
<td>Nested String, Sequence</td>
<td>Graph, Sequence</td>
</tr>
<tr>
<td>Main application</td>
<td>Syntax</td>
<td>“Semantics”</td>
</tr>
<tr>
<td></td>
<td>Document content</td>
<td>Meta Data</td>
</tr>
<tr>
<td>Schema</td>
<td>Complex content models (Alternative, Sequence, Repetition)</td>
<td>(Complex) Objecttypes</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Not existent</td>
<td>SubClass</td>
</tr>
<tr>
<td>Main application of schemas</td>
<td>Document Type Conformance</td>
<td>Vocabulary Specification</td>
</tr>
</tbody>
</table>

©2004/5, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis
Summary

• What is the difference between derived and associative metadata?

• How is an RDF statement structured?

• Which are the three views of RDF statements?

• What is reification?

• How is an RDF statement represented in XML?

• What can be specified with RDF schema?

• Which inheritance mechanisms does RDF schema provide?
3. Towards Ontology Languages

- Limitations of RDF Schema
  - Limited expressive power (subclass, property, subproperty)
  - Unclear semantics (e.g. subproperty never has been precisely defined)
  - No reasoning support

- Requirements on ontology languages
  - Well designed
    - Intuitive to human users
    - Adequate expressive power
  - Well defined
    - Clearly specified syntax (obviously)
    - Formal semantics (equally important)
    - Adequate expressive power
  - Compatible with existing (web) standards
    - in particular RDF

RDF has been recognized to have some clear deficiencies with respect to its use as ontology model. The set of basic modeling primitives is not very rich, even as compared e.g. to an ER model. Even worse, the semantics of RDF Schema is not well defined and it is therefore, for example, not clear when an RDF schema is consistent. For illustration consider the following example:

If P1 subPropertyOf P2 then we can conclude that in case that a class C1 has property P1, then C1 has also property P2. Assume now that the domain of P1 is C1, the domain of P2 is C2, and that in the schema C2 is declared to be a subclass of C1. From this it follows that only C2 can be domain of P2, which contradicts the fact that P1 is subproperty of P2. Since there exists no well-defined semantics of RDF, this kind of reasoning cannot be performed formally, and thus the meaning of such schemas remains unclear. Thus an ontology language is desired satisfying the following three requirements:

- It must be highly intuitive for the human user. For example, the object-oriented modeling paradigm is highly successful because of its intuitive nature, thus an ontology language should achieve the same. This must however not be at the cost of sacrificing the expressive power.

- It must have a well-defined formal semantics with established reasoning properties to ensure completeness, correctness, and efficiency in order to avoid problems as mentioned in the example before.

- It must have a proper linkage with existing Web languages such as XML and RDF to ensure interoperability.
OIL matches these criteria and unifies the three important aspects that different communities provide: rich modeling primitives as provided by the frame community (knowledge-based systems), formal semantics and efficient reasoning support as provided by description logics (a decidable fragment of first order logic), and a standard proposal for syntactical exchange notations as provided by the Web community.

- **Description Logics** describes knowledge in terms of concepts and role restrictions that are used to automatically derive classification taxonomies. They provide theories and systems for expressing structured knowledge, for accessing it and reasoning with it. OIL inherits from DL its formal semantics and the efficient reasoning support.

- **Frame-based systems** provide as central modeling primitive classes (i.e., frames) with attributes. These attributes do not have a global scope but are only applicable to the classes they are defined for. A frame provides a certain context for modeling one aspect of a domain. OIL incorporates the essential modeling primitives of frame-based systems.

- **Web standards: XML and RDF.** We must formulate a syntax of an ontology exchange language with existing Web standards for information representation. First, OIL has a well-defined syntax in XML based on a document type definition. Second, OIL is an extension of RDF and RDFS.
OWL as RDF Extension

- The language to specify OWL models is given as an RDF Schema
  - similarly as the RDF schema language is given within an RDF schema
  - therefore OWL models can be expressed in RDF
  - some of the OWL modeling primitives and RDF modeling primitives overlap and are re-used in OWL

```xml
<owl:Class rdf:ID="herbivore">
  <rdf:type
    rdf:resource="http://www.ontoknowledge.org/#DefinedClass"/>
  <rdfs:subClassOf rdf:resource="#animal"/>
  <owl:disjointWith rdf:resource="#carnivore"/>
</owl:Class>
```

This is an example of OWL as it is expressed in RDF. We will see that there exist also a native OWL language which abstracts from the RDF encoding.

In OWL as in RDFS classes are defined to categorize resources and classes can be subclasses of other classes. The class definition for "herbivore" gives however an example of the richer expressivity of OWL. One has the possibility to declare classes as subclasses of sets that are defined by complex so-called class expression, which are arbitrary combinations of other classes formed through the basic set operations of intersection, union and complement.

Without going further into further details of OWL this example should illustrate that OWL is in fact a richer model than RDFS.

Defining an ontology language as an extension of RDFS means that every RDFS ontology is a valid ontology in OWL. As a consequence, for example, an OWL processor will also understand RDFS. However, the other direction is also possible: Defining an OWL extension as closely as possible to RDFS allows maximal reuse of existing RDFS-based applications and tools. However, because the ontology language usually contains new primitives (and therefore a new vocabulary, which an RDFS processor does not know), 100 percent compatibility is impossible.

Let's look at the example. The OWL expression defines herbivore as a OWL class (a subclass of RDF:class), which is a subclass of animal and disjoint to all carnivores:

An application limited to pure RDFS can still capture some aspects of this definition:
OWL Semantics

• The semantics of OWL is given in Description Logics (DL)
  - Description Logics is a fragment (a sublanguage) of first order predicate logic
  - reasoning (deriving/prooﬁng statements) can be done (more) efﬁciently than in
    ﬁrst order predicate logic (FOL)

• Example

<table>
<thead>
<tr>
<th>OWL expression</th>
<th>disjointWith herbivore, carnivore</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL expression</td>
<td>herbivore ⊆ ¬ carnivore</td>
</tr>
<tr>
<td>FOL expression</td>
<td>∀x(herbivore(x) → ¬ carnivore(x))</td>
</tr>
</tbody>
</table>

• Using FOL reasoning (and thus DL reasoning) we could obtain, for example

  if herbivore ⊆ ¬ carnivore, and lion ⊆ carnivore
  then lion ⊆ ¬ herbivore

A crucial aspect of OWL is its formal semantics. The formal
semantics of OWL is given in terms of Description Logics. Without
discussing description logics in detail, it is possible to understand the
principle:

Description Logic is a fragment of ﬁrst order logic (FOL) that has
been developed with two things in mind: ﬁrst, it should allow to
reason about classes of objects and their relationships, and, second,
reasoning in description logic should be decidable and efﬁcient, i.e.
we should be able to provide an algorithm, that decides for every
statement that we make in DL whether it is true or false.

DL is exactly as OWL (and RDF) based on two basic constructs:
classes and properties. The syntax of DL is an abbreviated syntax of
FOL, which is for the non-expert awkward to read at the beginning.

We illustrate ﬁrst by an example of how classes are represented in
DL and then correspondingly also in FOL (which allows us to
properly understand the interpretation of OWL statements).

In this example we look at the OWL expression that was used to state
the condition that herbivores and carnivores are disjoint classes. The
 corresponding expression in DL is given below, and with some
imagination one can interpret it properly. For a precise interpretation
we give the corresponding expression in ﬁrst order logic below.

Since we know which are the laws for reasoning in ﬁrst order logic,
we can see immediately that the reasoning given below in
description logic is in fact correct.
Property Semantics in OWL

- **Classes in OWL (RDF) are unary predicates**
- **Slot constraints (= properties) in OWL are binary predicates**

<table>
<thead>
<tr>
<th>OWL expression</th>
<th>DL expression</th>
<th>FOL expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>class herbivore</code></td>
<td><code>herbivore ⊆ animal, ∀ eats.plant</code></td>
<td><code>∀x(herbivore (x) ↔ ∀y(eats(x, y) → plant(y)))</code></td>
</tr>
<tr>
<td><code>subClassOf animal</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>restriction onProperty eats allValuesFrom plant</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The second basic concept in DL besides classes are properties. We again give an example of how a property definition in the native OWL representation is represented as a description logic expression and the corresponding FOL expression. One observes that properties are represented in FOL as binary predicates. This should be obvious, if we consider that properties where used to connect resources (and where represented as edges of a graph in RDF)
This list provides an overview of all possible class constructors available in OWL.

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>$C_1 \cap \ldots \cap C_n$</td>
<td>Human $\cap$ Male</td>
</tr>
<tr>
<td>unionOf</td>
<td>$C_1 \cup \ldots \cup C_n$</td>
<td>Doctor $\sqcup$ Lawyer</td>
</tr>
<tr>
<td>complementOf</td>
<td>$\neg C$</td>
<td>$\neg$ Male</td>
</tr>
<tr>
<td>oneOf</td>
<td>${x_1} \cup \ldots \cup {x_n}$</td>
<td>${john} \cup {mary}$</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>$\forall P.C$</td>
<td>$\forall$ hasChild.Doctor</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>$\exists P.C$</td>
<td>$\exists$ hasChild.Lawyer</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>$\leq nP$</td>
<td>$\leq 1$ hasChild</td>
</tr>
<tr>
<td>minCardinality</td>
<td>$\geq nP$</td>
<td>$\geq 2$ hasChild</td>
</tr>
</tbody>
</table>
**OWL Axioms (Predicates)**

<table>
<thead>
<tr>
<th>Axiom</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>subClassOf</td>
<td>$C_1 \sqsubseteq C_2$</td>
<td>Human $\sqsubseteq$ Animal $\sqcup$ Biped</td>
</tr>
<tr>
<td>equivalentClass</td>
<td>$C_1 \equiv C_2$</td>
<td>Man $\equiv$ Human $\sqcap$ Male</td>
</tr>
<tr>
<td>disjointWith</td>
<td>$C_1 \sqsubseteq \neg C_2$</td>
<td>Male $\sqsubseteq \neg$ Female</td>
</tr>
<tr>
<td>sameIndividualAs</td>
<td>${x_1} = {x_2}$</td>
<td>${\text{President Rush}} = {G. W. Rush}$</td>
</tr>
<tr>
<td>differentFrom</td>
<td>${x_1} \subset \neg {x_2}$</td>
<td>${\text{John}} \subseteq \neg {\text{peter}}$</td>
</tr>
<tr>
<td>subPropertyOf</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>hasDaughter $\sqsubseteq$ hasChild</td>
</tr>
<tr>
<td>equivalentProperty</td>
<td>$P_1 = P_2$</td>
<td>cost $=$ price</td>
</tr>
<tr>
<td>inverseOf</td>
<td>$P_1 = P_2^\bot$</td>
<td>hasChild $=$ hasParent$^-$</td>
</tr>
<tr>
<td>transitiveProperty</td>
<td>$P^+ \sqsubseteq P$</td>
<td>ancestor$^+$ $\sqsubseteq$ ancestor</td>
</tr>
<tr>
<td>functionalProperty</td>
<td>$\top \sqsubseteq \bot_P$</td>
<td>$\top \subseteq \bot \text{hasMother}$</td>
</tr>
<tr>
<td>inverseFunctionalProperty</td>
<td>$T \sqsubseteq \bot_P^-$</td>
<td>$T \subseteq \bot \text{hasSSN}$</td>
</tr>
</tbody>
</table>

These are examples of predicates (used to state axioms) that can be given in OWL.
OWL sublanguages

- **OWL lite**
  - Subset of OWL (Full)
  - Easy to implement
- **OWL DL**
  - Restriction to FOL fragment
  - Well-defined semantics (Description Logic)
  - Mixing of RDFS and OWL restricted
  - Disjointness of classes, properties, individuals and data values
  - Well-defined computational properties
- **OWL Full**
  - Union of OWL syntax and RDF
  - No restrictions

OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full. OWL Lite was designed for easy implementation and to provide users with a functional subset that will get them started in the use of OWL. OWL DL (where DL stands for "Description Logic") was designed to support the existing Description Logic business segment and to provide a language subset that has desirable computational properties for reasoning systems. The complete OWL language (called OWL Full to distinguish it from the subsets) relaxes some of the constraints on OWL DL so as to make available features which may be of use to many database and knowledge representation systems, but which violate the constraints of Description Logic reasoners.
Summary

• Which are the reasons that RDF is not considered as being suitable as an ontology language?

• Which are the requirements an ontology language should satisfy?

• Which are examples of extensions in expressive power OWL introduces when compared to RDFS?

• Which approach is used to provide a precise semantics for OWL (and thus implicitly also for RDF)?
References

• Website
  - RDF: http://www.w3.org/
  - OWL: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.1

• Articles