RDF Meta Model and Schema

CS 431 - March 31, 2008
Carl Lagoze - Cornell University
Looking behind the curtain: RDF Meta-model
RDF Meta-Model provides base level for inferences

- Given a set of facts...
- Derive additional facts

- Some facts
  - Sam has a Prius
  - A Prius is a car
  - A Car is a type of vehicle
  - Sam has a bicycle
  - A bicycle is a type of vehicle

- Inference by subsumption: Sam has two vehicles
- Inference by human judgment: Sam is an environmentalist.
RDF meta-model basic elements

- All defined in rdf namespace
  - http://www.w3.org/1999/02/22-rdf-syntax-ns#
- Types (or classes)
  - rdf:Resource - everything that can be identified (with a URI)
  - rdf:Property - specialization of a resource expressing a binary relation between two resources
  - rdf:statement - a triple with properties rdf:subject, rdf:predicate, rdf:object
- Properties
  - rdf:type - subject is an instance of that category or class defined by the value
Use of rdf:type

• “Resource named http://foo.org/inst is member of class http://foo.org/classes/cl1”
• <http://foo.org/inst> <rdf:type> <http://foo.org/classes/cl1>
Typing the Resources in Statements

```xml
<?xml version="1.0" ?>
<rdf:RDF
  xmlns:gss="http://www.w3.org/2001/11/IsaViz/graphstylesheets#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ex="http://example.org/terms#">
  <ex:person rdf:about="info:123">
  </ex:person>
</rdf:RDF>
```
Formalizing a statement

• An RDF statement is a triple consisting of:
  - subject \( \rightarrow \) rdf:type resource
  - property \( \rightarrow \) rdf:type property
  - object \( \rightarrow \) rdf:type resource | literal
  - Examples

• Expressible as:
  - triple (ns1:s ns2:p ns3:o)
RDF statements and basic types

- RDF statements
  - rdf:statement

- Basic types
  - rdf:subject
  - rdf:property
  - rdf:object
Simple type inferencing

<table>
<thead>
<tr>
<th>explicit triple</th>
<th>Allows inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>(:s :p :o)</td>
<td>(:s rdf:type rdf:Resource)</td>
</tr>
<tr>
<td></td>
<td>(:p rdf:type rdf:Property)</td>
</tr>
<tr>
<td></td>
<td>(:o rdf:type rdf:Resource)</td>
</tr>
</tbody>
</table>
Reification - Statements about statements

"CL says 'WYA wrote Digital Libraries'"
Reification Structure

Staff member 85740 said the weight of item 10245 is 2.4 units
Reification XML

```xml
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:externs="http://www.example.com/terms/"
         xmlns:base="http://www.example.com/2002/04/products">
  <rdf:Description rdf:ID="item10245">
    <externs:weight rdf:datatype="&xsd;decimal">2.4</externs:weight>
  </rdf:Description>

  <rdf:Statement rdf:about="#triple12345">
    <rdf:subject rdf:resource="http://www.example.com/2002/04/products#item10245"/>
    <rdf:predicate rdf:resource="http://www.example.com/terms/weight"/>
    <rdf:object rdf:datatype="&xsd;decimal">2.4</rdf:object>
    <dc:creator rdf:resource="http://www.example.com/staffid/85740"/>
  </rdf:Statement>
</rdf:RDF>
```
Why Schema (1)?

- Enables communities to share machine readable tokens and locally define human readable labels.
Why Schema (2)?
Relationships among vocabularies

dc:Creator
marc:100
ms:director
bib:Author
Why Schema(3)?
Relationships among vocabulary elements

URI:R

ms:director \(\text{isA}\) dc:Creator

“John Smith”

ms:director \(\text{ms:director}\)

dc:Creator \(\text{dc:Creator}\)
RDF Schemas

- Declaration of vocabularies
  - classes, properties, and relationships defined by a particular community
  - relationship of properties to classes
- Provides substructure for inferences based on existing triples
- NOT prescriptive, but descriptive
  - NOTE: This is different from XML Schema
- Schema language is an expression of basic RDF model
  - uses meta-model constructs: resources, statements, properties
  - schema are “legal” rdf graphs and can be expressed in RDF/XML syntax
RDFs Namespace

• Class-related
  - rdfs:Class, rdfs:subClassOf

• Property-related
  - rdfs:subPropertyOf, rdfs:domain, rdfs:range
RDF Schema: Specializing Properties

- **rdfs:subPropertyOf** - allows specialization of relations
  - E.g., the property “father” is a subPropertyOf the property parent
- **subPropertyOf semantics**

<table>
<thead>
<tr>
<th>Explicit Model</th>
<th>Inferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>(:s rdfs:subPropertyOf :o)</td>
<td>(:s rdf:type rdf:Property) (:o rdf:type rdf:Property)</td>
</tr>
<tr>
<td>(:s :p :o) (:p rdfs:subPropertyOf :q)</td>
<td>(:s :q :o)</td>
</tr>
<tr>
<td>(:p rdfs:subPropertyOf :q) (:q rdfs:subPropertyOf :r)</td>
<td>(:p rdfs:subPropertyOf :r)</td>
</tr>
</tbody>
</table>
Inferences from Property Relationships

```graph
[Diagram showing property relationships between various entities with arrows indicating sub-properties and sibling relationships.]
```
Sub-Property Semantics

- Note the inferences we can not make at this time:
  - E.g., transitivity, reflexivity
- But, just wait (OWL)
Property-based semantics

- Provide basis for type inference from properties
- NOT restrictive like xml schema constraints
- `rdfs:domain`
  - classes of resources that have a specific property
- `rdfs:range`
  - classes of resources that may be the value of a specific property

<table>
<thead>
<tr>
<th>Explicit Model</th>
<th>Inferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>(:s :p :o)</td>
<td>(:s rdf:type :t)</td>
</tr>
<tr>
<td>(:p rdfs:domain :t)</td>
<td></td>
</tr>
<tr>
<td>(:s :p :o)</td>
<td>(:o rdf:type :t)</td>
</tr>
<tr>
<td>(:p rdfs:range :t)</td>
<td></td>
</tr>
</tbody>
</table>
Inferences from Constraints

`:has-child rdfs:domain parent`
`:has-child rdfs:range person`
`:has-sibling rdfs:domain person`
`:has-brother rdfs:range :male-person`
`:has-sister rdfs:range :female-person`
Class Declaration

- **rdfs:Class**
  - A resources denoting a set of resources;
  - Range of **rdf:type**

ex:MotorVehicle rdf:type rdfs:Class
exthings:companyCar rdf:type ex:MotorVehicle
Class Hierarchy

- rdfs:subClassOf
  - Create class hierarchy

```
ex:MotorVehicle rdf:type rdfs:Class
ex:SUV rdf:type rdfs:Class
ex:SUV rdf:subClassOf ex:MotorVehicle
exthings:companyCar rdf:type ex:SUV```

```
## Sub-Class Inferencing

<table>
<thead>
<tr>
<th>Explicit Model</th>
<th>Inferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>(:s rdf:type :o)</td>
<td>(:o rdf:type rdfs:Class)</td>
</tr>
<tr>
<td>(:s rdf:type :o) (:o rdfs:subClassOf :c)</td>
<td>(:s rdf:type :c)</td>
</tr>
<tr>
<td>(:s rdfs:subClassOf :o) (:o rdfs:subClassOf :c)</td>
<td>(:s rdfs:subClassOf :c)</td>
</tr>
<tr>
<td>(:s rdfs:subClassOf :o) (:o rdf:type rdfs:Class)</td>
<td>(:s rdf:type rdfs:Class)</td>
</tr>
<tr>
<td>(:s rdf:type rdfs:Class)</td>
<td>(:s rdfs:subClassOf rdf:Resource)</td>
</tr>
</tbody>
</table>
Sub-class Inferencing Example

(:parent rdfs:subClassOf :person)
(:male-person rdfs:subClassOf :person)
(:female-person rdfs:subClassOf :person)
(:mother rdfs:subClassOf :parent)
(:mother rdfs:subClassOf :female-person)
Components of the Semantic Web
Problems with RDF/RDFS
Non-standard, overly “liberal” semantics

• No distinction between class and instances
  - <Species, type, Class>
  - <Lion, type, Species>
  - <Leo, type, Lion>

• Properties themselves can have properties
  - <hasDaughter, subPropertyOf, hasChild>
  - <hasDaughter, type, Property>

• No distinction between language constructors and ontology vocabulary, so constructors can be applied to themselves/each other
  - <type, range, Class>
  - <Property, type, Class>
  - <type, subPropertyOf, subClassOf>

• No known reasoners for these non-standard semantics
Problems with RDF/RDFs
Weaknesses in expressivity

- **No localized domain and range constraints**
  - Can’t say the range of hasChild is person in context of persons and elephants in context of elephants

- **No existence/cardinality constraints**
  - Can’t say that all instances of persons have a mother that is also a person
  - Can’t say that persons have exactly two biological parents

- **No transitive, inverse or symmetric properties**
  - Can’t say isPartOf is a transitive property
  - Can’t say isPartOf is inverse of hasPart
  - Can’t say touches is symmetric
So, we need a more expressive and well-grounded ontology language....
What is an **Ontology**?

- A formal specification of conceptualization shared in a community
- Vocabulary for defining a set of things that exist in a world view
- Formalization allows communication across application systems and extension
- Parallel concepts in other areas:
  - **Domains**: database theory
  - **Types**: AI
  - **Classes**: OO systems
  - **Types/Sorts**: Logic
- **Global vs. Domain-specific**
XML and RDF are **ontologically neutral**

- **No standard vocabulary just primitives**
  - Resource, Class, Property, Statement, etc.

- **Compare to classic first order logic**
  - Conjunction, disjunction, implication, existential, universal quantifier
Components of an Ontology

• Vocabulary (concepts)
• Structure (attributes of concepts and hierarchy)
• Relationships between concepts
• Logical characteristics of relationships
  - Domain and range restrictions
  - Properties of relations (symmetry, transitivity)
  - Cardinality of relations
  - etc.
Wordnet

- On-line lexical reference system, domain-independent
- >100,000 word meanings organized in a taxonomy with semantic relationships
  - Synonymy, meronymy, hyponymy, hypernymy
- Useful for text retrieval, etc.
CYC

- Effort in AI community to accommodate all of human knowledge!!!
- Formalizes concepts with logical axioms specifying constraints on objects and classes
- Associated reasoning tools
- Contents are proprietary but there is OpenCyc
  - http://www.opencyc.org/
So why re-invent ontologies for the Web

• Not re-invention
  - Same underlying formalisms (frames, slots, description logic)
• But new factors
  - Massive scale
    • Tractability
    • Knowledge expressiveness must be limited or reasoning must be incomplete
  - Lack of central control
    • Need for federation
    • Inconsistency, lies, re-interpretations, duplications
    • New facts appear and modify constantly
  - Open world vs. Close world assumptions
    • Contrast to most reasoning systems that assume anything absent from knowledge base is not true
    • Need to maintain monotonicity with tolerance for contradictions
• Need to build on existing standards
  • URI, XML, RDF
Web Ontology Language (OWL)

- W3C Web Ontology Working Group (WebOnt)
- Follow on to DAML, OIL efforts
- W3C Recommendation
- Vocabulary extension of RDF
Species of OWL

• **OWL Lite**
  - Good for classification hierarchies with simple constraints (e.g., thesauri)
  - Reasoning is computational simple and efficient

• **OWL DL**
  - Computationally complete and decidable (computation in finite time)
  - Correspondence to *description logics* (decidable fragment of first-order logic)

• **OWL Full**
  - Maximum expressiveness
  - No computational guarantees (probably never will be)

• Each language is extension of simpler predecessor
Description Logics

- Fragment of first-order logic designed for logical representation of object-oriented formalisms
  - frames/classes/concepts
    - sets of objects
  - roles/properties
    - binary relations on objects
  - individuals
- Representation as a collection of statements, with unary and binary predicates that stand for concepts and roles, from which deductions can be made
- High expressivity with decidability and completeness
  - Decidable fragment of FOL
Description Logics Primitives

- **Atomic Concept**
  - Human

- **Atomic Role**
  - likes

- **Conjunction**
  - human intersection male

- **Disjunction**
  - nice union rich

- **Negation**
  - not rich

- **Existential Restriction**
  - exists has-child.Human

- **Value Restriction**
  - for-all has-child.Blond

- **Number Restriction**
  - ≥ 2 has-wheels

- **Inverse Role**
  - has-child, has-parent

- **Transitive role**
  - has-child
Description Logic - Tboxes

• Terminological knowledge
• Concept Definitions
  - Father is conjunction of Man and has-child.Human
• Axioms
  - motorcycle subset-of vehicle
  - has-favorite.Brewery subrelation-of drinks.Beer
Description Logics: Aboxes

- **Assertional knowledge**
- **Concept assertions**
  - John is-a Man
- **Role assertions**
  - has-child(John, Bill)
Description Logics: Basic Inferencing

- **Subsumption**
  - Is C1 subclass-of C2
  - Compute taxonomy

- **Consistency**
  - Can C have any individuals
Namespaces and OWL

<rdf:RDF
    xmlns="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xmlns:vin="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xml:base="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"/>
OWL Class Definition

<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>

<owl:Class rdf:ID="Wine">
   <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
   <rdfs:label xml:lang="en">wine</rdfs:label>
   <rdfs:label xml:lang="fr">vin</rdfs:label>
   ...
</owl:Class>
Why owl:class vs. rdfs:class

• Rdfs:class is “class of all classes”
• In DL class can not be treated as individuals (undecidable)
• Thus owl:class, which is expressed as rdfs:subclass of rdfs:class
  - No problem for standard rdf processors since an owl:class “is a” rdfs:class

• Note: there are other times you want to treat class of individuals
  - Class drinkable liquids has instances wine, beer, ....
  - Class wine has instances merlot, chardonnay, zinfandel, ...
OWL class building operations

- **disjointWith**
  - No vegetarians are carnivores
- **sameClassAs** (equivalence)
- **Enumerations** (on instances)
  - The Ivy League is Cornell, Harvard, Yale, ....
- **Boolean set semantics** (on classes)
  - Union (logical disjunction)
    - Class `parent` is union of `mother`, `father`
  - Intersection (logical conjunction of class with properties)
    - Class `WhiteWine` is conjunction of things of class `wine` and have property `white`
  - complimentOf (logical negation)
    - Class `vegetarian` is disjunct of class `carnivore`
OWL Properties

Two types
- **ObjectProperty** - relations between instances of classes
- **DatatypeProperty** - relates an instance to an `rdfs:Literal` or XML Schema datatype

(Both `rdfs:subClassOf rdf:Property`)

```xml
<owl:DatatypeProperty rdf:ID="name">
    <rdfs:domain rdf:resource="Person" />
    <rdfs:range rdf:resource= "http://www.w3.org/2001/XMLSchema/string" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="activity">
    <rdfs:domain rdf:resource="Person" />
    <rdfs:range rdf:resource="ActivityArea" />
</owl:ObjectProperty>
```
OWL property building operations & restrictions

• Transitive Property
  - \( P(x,y) \) and \( P(y,z) \) -> \( P(x,z) \)

• Symmetric Property
  - \( P(x,y) \) iff \( P(y,x) \)

• Functional Property
  - \( P(x,y) \) and \( P(x,z) \) -> \( y=z \)

• inverseOf
  - \( P1(x,y) \) iff \( P2(y,x) \)

• InverseFunctional Property
  - \( P(y,x) \) and \( P(z,x) \) -> \( y=z \)

• Cardinality
  - Only 0 or 1 in lite and full
OWL DataTypes

• Full use of XML schema data type definitions
• Examples
  - Define a type age that must be a non-negative integer
  - Define a type clothing size that is an enumeration “small” “medium” “large”
OWL Instance Creation

• Create individual objects filling in slot/attribute/property definitions

<Person ref:ID="William Arms">
  <rdfs:label>Bill</rdfs:label>
  <age><xsd:integer rdf:value="57"/></age>
  <shoesize><xsd:decimal rdf:value="10.5"/></shoesize>
</Person>
**OWL Lite Summary**

Schema constructs
- Class (i.e. owl:Class)
- rdf:Property
- rdfs:subClassOf
- rdfs:subPropertyOf
- rdfs:domain
- rdfs:range
- Individual

Property characteristics
- inverseOf
- TransitiveProperty
- FunctionalProperty
- InverseFunctionalProperty
- SymmetricProperty

Equality constructs
- equivalentClass
- equivalentProperty
- sameIndividualAs
- differentFrom
- allDifferent

Cardinality
- minCardinality
- maxCardinality
- Cardinality (0 or 1)

Class intersection
- intersectionOf

Headers
- imports
- priorVersion
- backwardCompatibleWith
- incompatibleWith

Property type restrictions
- allValuesFrom
- someValuesFrom

RDF datatyping
### OWL DL and Full Summary

<table>
<thead>
<tr>
<th>Class axioms</th>
<th>Class expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>oneOf</td>
<td>equivalentClass</td>
</tr>
<tr>
<td>disjointWith</td>
<td>rdfs:subClassOf</td>
</tr>
<tr>
<td></td>
<td>unionOf</td>
</tr>
<tr>
<td></td>
<td>intersectionOf</td>
</tr>
<tr>
<td></td>
<td>complementOf</td>
</tr>
<tr>
<td>Property fillers</td>
<td>Arbitrary cardinality</td>
</tr>
<tr>
<td>hasValue</td>
<td>minCardinality</td>
</tr>
<tr>
<td></td>
<td>maxCardinality</td>
</tr>
<tr>
<td></td>
<td>Cardinality</td>
</tr>
</tbody>
</table>
OWL DL vs. OWL-Full

• Same vocabulary

• OWL DL restrictions
  - Type separation
    • Class can not also be an individual or property
    • Property can not also be an individual or class
  - Separation of ObjectProperties and DatatypeProperties
## Language Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>DTD</th>
<th>XSD</th>
<th>RDF(S)</th>
<th>OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounded lists (&quot;X is known to have exactly 5 children&quot;)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Cardinality constraints (Kleene operators)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class expressions (unionOf, complementOf)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Data types</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Enumerations</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalence (properties, classes, instances)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Formal semantics (model-theoretic &amp; axiomatic)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Inheritance</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Inference (transitivity, inverse)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Qualified constraints (&quot;all children are of type person&quot;)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Reification</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Protégé and RACER - tools for building, manipulating and reasoning over ontologies

• Protégé - http://protege.stanford.edu/
  - Use the 3.x version
  - Multiple plug-ins are available

• Protégé OWL plug-in
  - http://protege.stanford.edu/plugins/owl/

• Other semantic web related plug-ins
  - http://protege.cim3.net/cgi-bin/wiki.pl?ProtegePluginsLibraryByTopic#nid349

• Racer
  - Description Logic based reasoning engine
  - Server-based
  - Integrates with Protégé-OWL