
CmpE 596: Service-Oriented Computing

Pinar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Bogazici University

- Resource Description Framework (RDF)
- Provides a basis for knowledge representation
- Simple language to capture assertions (statements), which help capture knowledge, e.g., about resources on the Web
- Describe the properties and property values
- Allow machines to interpret resource descriptions
- Resources: Web page; person; book

Why RDF?

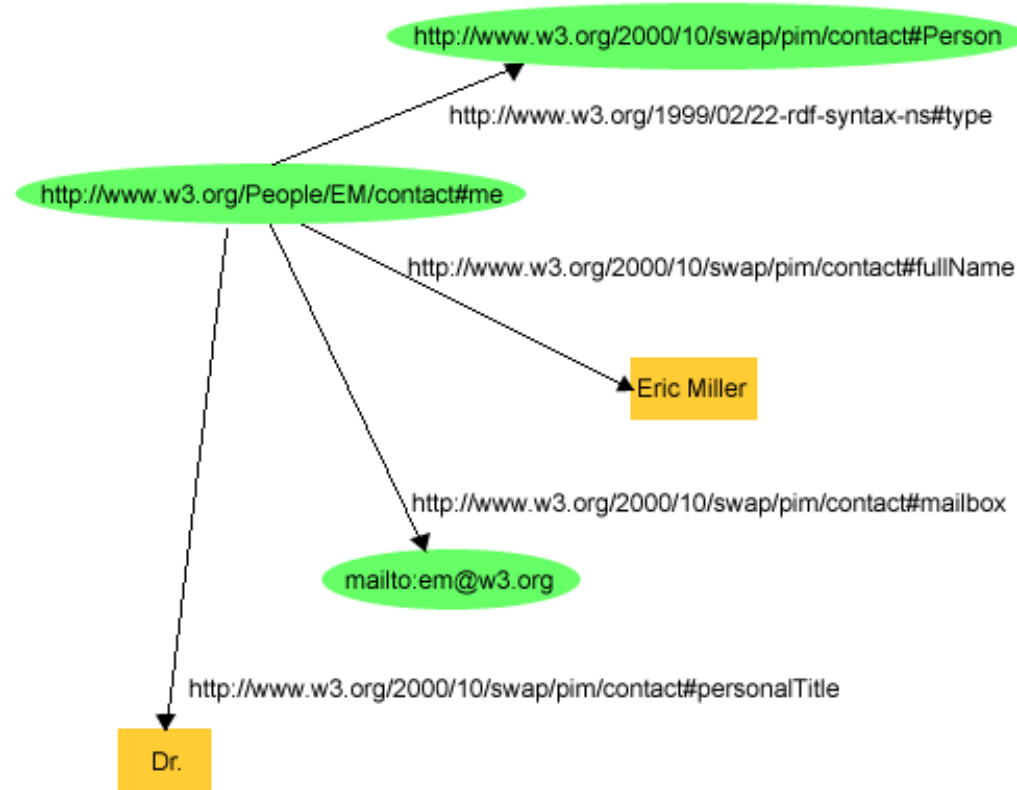
- XML
 - Gives us a document tree
 - Doesn't identify the content represented by a document
 - Enables multiple representations for the same content
- RDF expresses the content itself

Basic Concepts

- Describe things that have properties which have values
- Triples of <subject, predicate, object>
- Ex: <Eric Miller, hasTitle, Dr.>
- Need to uniquely identify each so that they can be processed without confusion

RDF Graph

- Two nodes for the subject and object
- An arc for the predicate
- The resource identified by URI
<http://www.w3.org/People/EM> is of type Person, whose full name is Eric Miller, whose mailbox is em@w3.org and whose personal title is Dr.
- Ovals are URIs; rectangles are literals (only as objects)



RDF Syntax

- Equivalent XML-based Syntax (RDF/XML)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
xmlns:contact="http://www.w3.org/2000/10/swap/pim/conta
ct#">
  <contact:Person
rdf:about="http://www.w3.org/People/EM/contact#me">
  <contact:fullName>Eric Miller</contact:fullName>
  <contact:mailbox rdf:resource="mailto:em@w3.org"/>
  <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

RDF Containers

- Containers
 - `rdf:Bag` (no order; may include duplicates)
 - `rdf:Seq` (order important; may include duplicates)
 - `rdf:Alt` (list of alternatives)
- Container resource (parent); members (child)
- Membership properties names are enumerated as `rdf:_1`, `rdf:_2`, ..., `rdf:_n`
- Semantics vs. Intended Meaning
 - RDF only intends that members of `rdf:Alt` will be specified and interpreted as alternatives.
 - RDF cannot and does not check that they are really alternatives or not.

RDF Collections

- Collections
 - Contains members (like Containers)
 - Implies that the listed members are the entire collection
- Predefined collection vocabulary (similar to Lisp lists)
 - `rdf:List` (entire list)
 - `rdf:first` (the first element of a given list)
 - `rdf:rest` (take the first element out; the remaining list)
 - `rdf:nil` (a list without any elements)

RDF Reification (1)

- RDF statements about other RDF statement
 - Meta information about RDF
 - Who wrote this RDF? The date of creation, etc.
- RDF Reification Vocabulary
 - `rdf:Statement`
 - `rdf:subject`
 - `rdf:predicate`
 - `rdf:object`
- Example:

```
exproducts:triple12345    rdf:type    rdf:Statement .
exproducts:triple12345    rdf:subject  exproducts:item10245 .
exproducts:triple12345    rdf:predicate exterms:weight .
exproducts:triple12345    rdf:object   "2.4"^^xsd:decimal .
```

RDF Reification (2)

```
<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>
```

- Staff 85740 stated that item 10245 weights 2.4 (kg).
- We don't really know the unit; but a unit is necessary!

RDF Schema

- Used to describe the vocabulary for resources
- Examples: `exterms:Tent`; `ex2:Book`
- RDF Schema provides the types that will be used in RDF statements
- RDF talks about objects; RDF schema defines classes for objects

RDF Schema Classes

- Classes have a property *rdf:type* and a value *rdfs:Class*
 - `externs:Tent rdf:type rdfs:Class`
 - `externs:outdoorsTent rdf:type rdfs:Class`
- Resources can be defined based on new classes
 - `externs:myGreenTent rdf:type externs:Tent`
- Classes can be specialized using *rdfs:subClassOf* property
 - `externs:outdoorsTent rdfs:subClassOf externs:Tent`
 - `rdfs:Resource`

RDF Schema Properties (1)

- Classes have a properties denoted by *rdf:property*
ex:Person rdf:type rdfs:Class .
ex:author rdf:type rdf:Property .
- *rdfs:range* denotes the range of values for a property
Ex1: ex:author rdfs:range ex:Person .
Ex2: ex:age rdf:type rdf:Property .
ex:age rdfs:range xsd:integer .
xsd:integer rdf:type rdfs:Datatype .

RDF Schema Properties (2)

- *rdfs:domain* denotes the values for which a property applies.

Ex: `ex:Book rdf:type rdfs:Class .`
`ex:author rdf:type rdf:Property .`
`ex:author rdfs:domain ex:Book .`

- A property can have 0 or more domain and range.
- Properties can be specialized using `rdfs:subPropertyOf`

Ex: `ex:driver rdf:type rdf:Property .`
`ex:primaryDriver rdf:type rdf:Property .`
`ex:primaryDriver rdfs:subPropertyOf ex:driver .`

RDF Discussion (1)

- JAVA: Class *book* has an attribute *author* of type *person*
- RDF: There is an *author* property between a *book* and a *person*
- JAVA: If you are talking about a *newspaper*, you need to define a new *author* attribute (Local scope)
- RDF: Define an *author* property once. (Global scope)
- JAVA: You can't talk about an *author* attribute without a class
- RDF: You can if you don't specify a domain

RDF Discussion (2)

- JAVA:
 - Class *sportsarticle* has an attribute *author* of type *male*
 - Class *newsarticle* has an attribute *author* of type *female*
- RDF: Cannot match different domains with ranges
- JAVA is prescriptive
 - Won't allow a male as the author of a news article
- RDF is descriptive; usage is application-dependent
 - Enforce constraints (like JAVA)
 - If the author of a news article is not known infer female
 - Accept the existence of a news article without an author
 - Accept a news article with an *editor* attribute instead

RDF Discussion (3)

- What you cannot say in RDF:
 - Cardinality: Each book has one author.
 - Transitivity: A ex:hasAncestor B, and B ex:hasAncestor C, then A ex:hasAncestor C.
 - Class Equivalence: Two classes (with different URIs) are the same
 - Instance Equivalence: Two objects (with different URIs) are the same
 - Different cardinality requirements for the same property: A homework has one author; but a book may have several.
 - Classes based on others: Two classes are disjoint

Web Ontology Language

Based largely on

Service-Oriented Computing: Semantics, Processes, Agents

– Munindar P. Singh and Michael N. Huhns, Wiley, 2004

Web Ontology Language (OWL)

- RDF captures the basics, i.e., an object-oriented type system
- Additional subtleties of meaning are needed for effective KR
- OWL standardizes additional constructs to show how to capture such subtleties of meaning
- OWL builds on RDF

- Specifies classes and properties in a form of description logic (DL)
 - Class operators analogous to Boolean operators *and*, *not*, and *or*
 - Constraints on properties: transitive, ...
 - Restrictions: constructs unique to DL
- Has three species: OWL Full, OWL DL, and OWL Lite

Custom Metadata Vocabularies

- Custom vocabularies for such metadata
- The metadata must be given a standard semantics so that different parties interpret it the same way, and so that tools can function appropriately.

```
<Mammal rdf:ID="Mary"/>
```

```
<Mammal rdf:ID="John">
```

```
  <hasParent rdf:resource="#Mary"/>
```

```
</Mammal>
```

Vocabulary Semantics

- A trivial ontology defining our vocabulary
- Uses simple subclasses and properties
 - Disjointness goes beyond RDF
 - Object properties refine RDF properties; relate two objects

```
<owl:Class rdf:ID="Mammal">  
  <rdfs:subClassOf rdf:resource="#Animal"/>  
  <owl:disjointWith rdf:resource="#Reptile"/>  
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID="hasParent">  
  <rdfs:domain rdf:resource="#Animal"/>  
  <rdfs:range rdf:resource="#Animal"/>  
</owl:ObjectProperty>
```

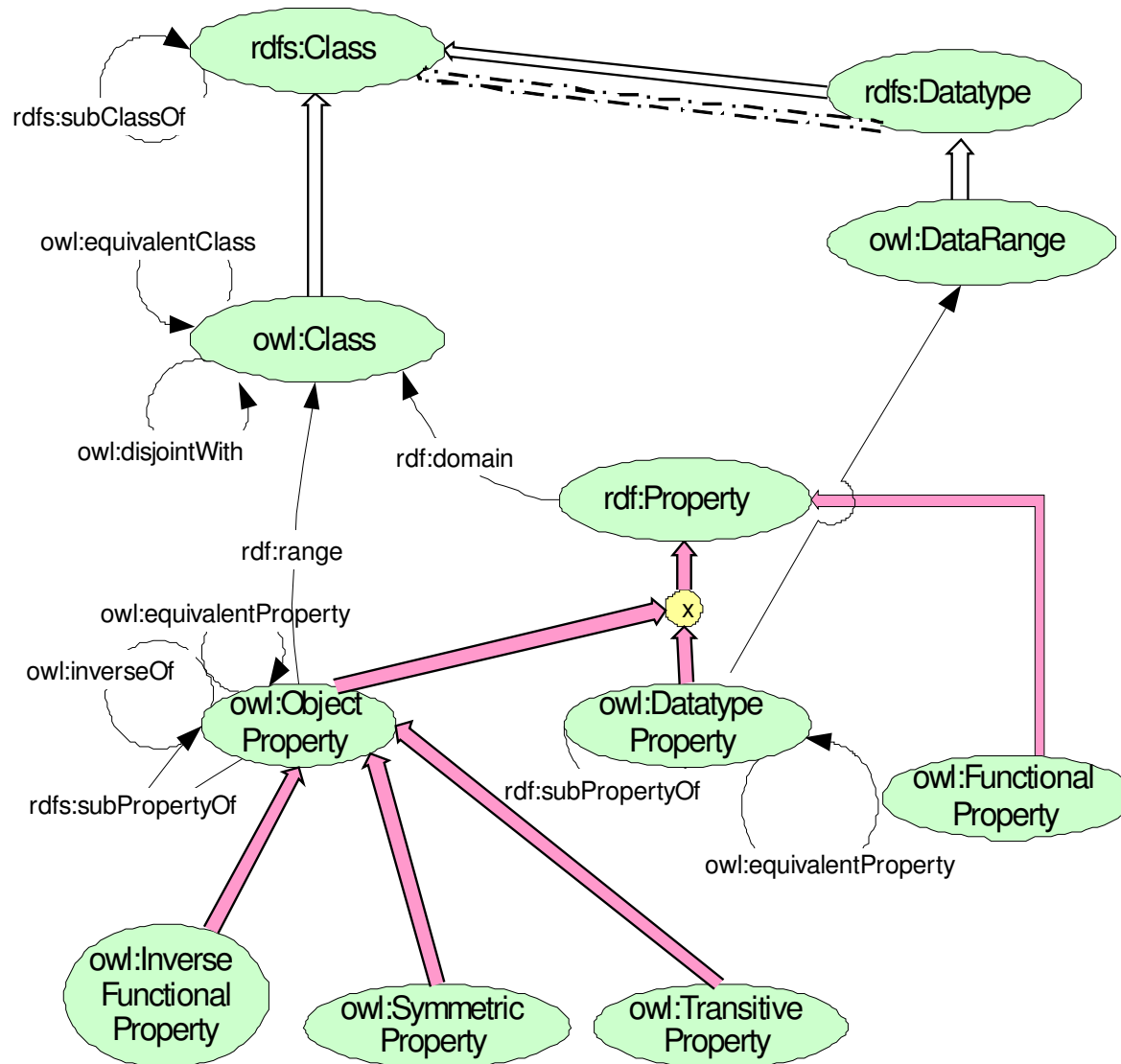
Simple Inference

- Given the definition for the property hasParent and

```
<owl:Thing rdf:ID="Fido">  
  <hasParent rdf:resource="#Rover"/>  
</owl:Thing>
```

we can infer that Fido is an Animal

OWL Entities and Relationships



OWL Object Properties

- Transitive Property
 - $P(x,y)$ and $P(y,z)$ implies $P(x, z)$

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="&owl;TransitiveProperty" /> <rdfs:domain
  rdf:resource="&owl;Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>
<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>
<Region rdf:ID="CaliforniaRegion">
  <locatedIn rdf:resource="#USRegion" />
</Region>
```

OWL Object Properties

- Symmetric Property
 - $P(x,y)$ iff $P(y,x)$

```
<owl:ObjectProperty rdf:ID="adjacentRegion">
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
  <rdfs:domain rdf:resource="#Region" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>
<Region rdf:ID="MendocinoRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
  <adjacentRegion rdf:resource="#SonomaRegion" />
</Region>
```

OWL Object Properties

- Functional Property
 - $P(x, y)$ and $P(x, z)$ implies $y = z$

```
<owl:Class rdf:ID="VintageYear" />
```

```
<owl:ObjectProperty rdf:ID="hasVintageYear">
```

```
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
```

```
  <rdfs:domain rdf:resource="#Vintage" />
```

```
  <rdfs:range rdf:resource="#VintageYear" />
```

```
</owl:ObjectProperty>
```

OWL Object Properties

- inverseOf
 - $P1(x,y)$ iff $P2(y,x)$

```
<owl:ObjectProperty rdf:ID="hasMaker">  
  <rdf:type rdf:resource="&owl;FunctionalProperty" />  
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="producesWine">  
  <owl:inverseOf rdf:resource="#hasMaker" />  
</owl:ObjectProperty>
```

OWL Object Properties

- inverseFunctionalProperty
 - $P(y,x)$ and $P(z,x)$ implies $y=z$

```
<owl:ObjectProperty rdf:ID="hasMaker" />
```

```
<owl:ObjectProperty rdf:ID="producesWine">
```

```
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
```

```
  <owl:inverseOf rdf:resource="#hasMaker" />
```

```
</owl:ObjectProperty>
```

Constructing OWL Classes

- Explicitly (as in the examples above)
or
- Anonymously, using
 - Restrictions
 - Set operations

Restrictions: 1

- A unique feature of description logics
- Like division: define classes in terms of a restriction that they satisfy with respect to a given property
- Anonymous: typically included in a class def to enable referring them
- Key primitives are
 - someValuesFrom a specified class
 - allValuesFrom a specified class
 - hasValue equal to a specified individual or data type
 - minCardinality
 - maxCardinality
 - Cardinality (when maxCardinality equals minCardinality)

Restrictions: 2

```
• <owl:Class rdf:ID="Wine">  
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />  
  <rdfs:subClassOf>  
    <owl:Restriction>  
      <owl:onProperty rdf:resource="#hasMaker" />  
      <owl:allValuesFrom rdf:resource="#Winery" />  
    </owl:Restriction>  
  </rdfs:subClassOf>  
  ...  
</owl:Class>
```

The maker of a Wine must be a Winery. The allValuesFrom restriction is on the hasMaker property of this Wine class *only*. Makers of Cheese are not constrained by this local restriction

Set operators

- intersectionOf, unionOf, complementOf

```
<owl:Class rdf:ID='SugaryBread'>
```

```
  <owl:intersectionOf rdf:parseType='Collection'>
```

```
    <owl:Class rdf:about='#Bread' />
```

```
    <owl:Class rdf:about='#SweetFood' />
```

```
  </owl:intersectionOf>
```

```
</owl:Class>
```

```
<owl:Class rdf:ID="Fruit">
```

```
  <rdfs:subClassOf rdf:resource="#SweetFruit" />
```

```
  <rdfs:subClassOf rdf:resource="#NonSweetFruit" />
```

```
</owl:Class>
```

- Assertions that are given to be true
- Can be especially powerful in combination with other axioms, which may come from different documents
- Some primitives
 - `rdfs:subClassOf`
 - `owl:equivalentClass`

Axioms: 2

```
<owl:AllDifferent> <!-- in essence, pair-wise inequalities -->  
  <owl:distinctMembers rdf:parseType='Collection'>  
    <ex:Country rdf:ID='Russia'/>  
    <ex:Country rdf:ID='India'/>  
    <ex:Country rdf:ID='USA'/>  
  </owl:distinctMembers/>  
</owl:AllDifferent>
```

```
<ex:Country rdf:ID='Iran'/>  
<ex:Country rdf:ID='Persia'>  
  <owl:sameIndividualAs rdf:resource='#Iran'/>  
</ex:Country>
```

Restrictions versus Axioms

- Axioms are global assertions that can be used as the basis for further inference
- Restrictions are constructors
 - When we state that hasFather has a maxCardinality of 1, we are
 - Defining the class of animals who have zero or one fathers: this class may or may not have any instances
 - Not stating that all animals have zero or one fathers
- Often, to achieve the desired effect, we would have to combine restrictions with axioms (such as based on equivalentClass)

Inference

- OWL is about content, not the syntax
- Statements from different documents about the same URI are automatically conjoined
- OWL declarations may seem conflicting
 - Declare that no one can have more than one mother
 - Declare Mary is John's mother
 - Declare Jane is John's mother
- A DBMS would declare an integrity violation
- An OWL reasoner would say $Mary = Jane$

Resources

- Wine ontology: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf>
- Food ontology: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/food.rdf>
- W3C: <http://www.w3.org/2004/OWL/>
- Editor: Protégé <http://protege.stanford.edu/>
- Reasoners:
 - Pellet: <http://www.mindswap.org/2003/pellet/index.shtml>
 - Jena: <http://jena.sourceforge.net/>

Expressiveness Limitations: 1

OWL DL cannot express some simple requirements

- *Non-tree models*: because instance variables are implicit in OWL restrictions, OWL cannot express conditions that require that two variables be identified
 - Think of siblings – two people who have the *same* parents – but in terms of classes
 - Do the same thing with class definitions

Expressiveness Limitations: 2

Specialized properties

- Cannot state that the child of a mammal must be a mammal and so on without
 - Defining new child properties for each class
 - Adding an axiom for each class stating that it is a subClassOf the restriction of hasChild to itself
- Analogous to the problem in a strongly typed object-oriented language without generics
 - You have to typecast the contents of a hash table or linked list

Expressiveness Limitations: 3

- Constraints among individuals
 - Cannot define tall person: class of persons whose height is above a certain threshold
 - Can define ETHusband: class of persons who have been married to Elizabeth Taylor
- Cannot capture defeasibility (also known as nonmonotonicity)
 - Birds fly
 - Penguins are birds
 - Penguins don't fly

Ontology Management

- Descriptions of services are improved through the use of ontologies
 - But how do we make sure the parties involved agree upon and understand the ontologies needed?
- Traditional approach: standardize the ontologies via a formal process
- Emerging approach: be more like the Web; figure out the “correct” ontology via consensus

Standard Ontologies

Standardization is more a sociopolitical than a technical process

- IEEE Standard Upper Ontology
- Common Logic (language and upper-level ontology)
- Process Specification Language
- Space and time ontologies
- Domain-specific ontologies, such as health care, taxation, shipping, ...