RDF, RDFS, & OWL

A 3-Hour Introductory Training Course

Mark Wallace
Principal Engineer, Semantic Applications
mwallace@modusoperandi.com
Overview

Level: Introductory
• No Prior RDF knowledge necessary
• Familiarity with XML and some basic Computer Science concepts a plus

• Duration: 3.5 hours, with breaks

What we will Cover:
• What is RDF?
  – Graph Model, Triples
  – Namespacing
  – Serialization Formats
  – RDF Querying
• What is RDFS?
  – How it adds to RDF
  – RDFS Constructs & Inference
• Introduction to OWL
  – How it adds to RDFS
  – How it is more restricted than RDF and RDFS
  – OWL Constructs & Inference
  – OWL Dialects
  – Open World Assumption
RDF
What is RDF?

• RDF = Resource Description Framework
• For making statements about web Resources
  – E.g.
  – http://modusoperandi.com/index.htm author "Judy Gasperini"
  – Or
RDF is a Graph Model

- Graphs have nodes and edges
- A directed graph: edges have one direction

http://modusoperandi.com/index.htm

http://modusoperandi.com/ont#authorOf

http://xmlns.com/foaf/0.1/workplaceHomepage

http://xmlns.com/foaf/0.1/authorOf

http://xmlns.com/foaf/0.1/name

http://xmlns.com/foaf/0.1/knows

"Judy Gasperini"

http://modusoperandi.com/ont#MarkW
Triples

• Graphs are nice, but how to easily represent?
• Answer: Triples!
• All statements are made as a three-valued statement: subject - predicate - object

• Exact same resource name indicates same graph node:
  – E.g.:
    – \([s1\ p1\ o1]\)
    – \([s1\ p2\ o2]\)
• All predicates are directional (a directed graph)
• Nodes are subjects and objects
• A triple is also called a statement or a fact
Exercise

• Draw the graph represented by these triples
  – s1 p1 o1
  – s1 p2 "hello"
  – s1 p1 o2
  – s2 p3 o2
Resources and Literals

• A Resource is a URI
  – URN or URL
  – E.g.
    • http://books.com/MobyDick
    • http://modusoperandi.com/ont#authorOf
    • urn:issn:1391-3793

• Literal is a non-URL string value
  – E.g.
    • "Judy Gasperini"
    • "100"
    • "3.1415926"
    • "true"
    • "book"@en (language code)
    • "libro"@es (language code)
    • "50"^^xsd:integer (XSD data type)
    • "2002-02-12"^^xsd:date (XSD data type)

• Subject and Predicate can only be resources (URIs)
• Object can be a resource or a literal
Exercise

• Which are literals? Which are URIs?
  – "12"
  – "Hello"
  – http://www.w3.org/2000/01/rdf-schema#subClassOf
  – "20090506T12:34:56"
  – urn:isbn:020161586X
  – "false"^^xsd:boolean

• Which statements are illegal? Why?
  – "1" urn:rdf:typeof http://purl.org/type#integer
  – urn:mo:ppl:JudyG urn:mo:fullName "Gasperini"
  – "Gasperini" http://w3.org/empl#worksAt "Modus Operandi"
  – urn:mo:ppl:JudiG "works at" urn:mo:MOHQ
N-Triples format

- One triple per line
- All URIs surrounded by angle brackets (<, >)
- All literals surrounded by double-quotes
- All lines terminated with period ".
- E.g.:

```
```

- Good for sorting and filtering!
- See [http://www.w3.org/TR/rdf-testcases/#ntriples](http://www.w3.org/TR/rdf-testcases/#ntriples)
Namespaces and QNames

• Namespaces are the common front-part of URLs
• E.g. MO owns the "modusoperandi.com" namespace, and anything under it
• XML allows a way of mapping namespaces to prefixes, e.g.
  xmlns:foaf="http://xmlns.com/foaf/0.1/" (ns prefix)

  – So we can then say...
    foaf:knows  (called a Qualified Name, or QName)

  – instead of...
    http://xmlns.com/foaf/0.1/knows
Serialization Format: RDF/XML

- XML format for RDF info. E.g.:

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:mo="http://modusoperandi.com/ont#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <rdf:Description rdf:about="http://modusoperandi.com/ont#JudyG">
    <foaf:name>Judi Gasperini</foaf:name>
    <foaf:knows>
      <rdf:Description rdf:about="http://modusoperandi.com/ont#MarkW">
      </rdf:Description>
    </foaf:knows>
  </rdf:Description>
</rdf:RDF>
```
RDF/XML "Striping"

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/
   xmlns:mo="http://modusoperandi.com/ont#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
   <rdf:Description rdf:about="http://modusoperandi.com/ont#JudyG">
     <foaf:name>Judi Gasperini</foaf:name>
     <foaf:knows>
       <rdf:Description rdf:about="http://modusoperandi.com/ont#MarkW">
       </rdf:Description>
     </foaf:knows>
   </rdf:Description>
</rdf:RDF>

Each Level alternates between Resource and Property
Serialization Format: Turtle

• Turtle = Terse RDF Triple Language
• http://www.w3.org/TeamSubmission/turtle/
• Compact, simpler than RDF/XML
• Superset of N-Triples
  – E.g.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix mo:  <http://modusoperandi.com/ont#> .

mo:JudyG foaf:name "Judi Gasperini"
  ; mo:authorOf <http://modusoperandi.com/index.htm>
  ; foaf:knows mo:MarkW
  .

mo:MarkW  foaf:workplaceHomepage

• Can use namespace prefixes
• Full URIs surrounded by angle brackets
• Ways to abbreviate groups of triples (semicolon, comma)
• Multiple lines allowed per triple
Exercise

• Draw the graph represented by:

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:mo="http://modusoperandi.com/ont#">
  <rdf:Description rdf:about="http://modusoperandi.com/ont#JudyG">
    <mo:hasPhone>321-555-1212</mo:hasPhone>
  </rdf:Description>
</rdf:RDF>
```
Exercise

• Draw the graph represented by:

@prefix mo: <http://modusoperandi.com/ont#> .

mo:JudyG mo:worksFor mo:ModusOperandi .
Querying RDF

- Query is via graph matching.
- E.g., a graph-matching pattern:
  - ?p rdf:type foaf:Person
  - ?p foaf:name ?n
- Meaning...
  - Match any part of the graph where a node (as "?p") has edge rdf:type to node foaf:Person, and that same node (?p) has edge foaf:name to any node (as "?n")
- All places in the RDF graph that match the pattern are returned by the query.
Querying RDF

• **SPARQL**
  – SPARQL Protocol and RDF Query Language
  – E.g.

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix foaf: <http://xmlns.com/foaf/0.1/>

select * where {
}
```

Result Set (from previous example):

<table>
<thead>
<tr>
<th>p</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://modusoperandi.com/ont#JudyG">http://modusoperandi.com/ont#JudyG</a></td>
<td>&quot;Judy Gasperini&quot;</td>
</tr>
</tbody>
</table>
Distributed Data Merging

• RDF triples can be spread across many files
• As long as resources have the exact same URI they mean the same thing
• Allows large RDF graphs to be broken into smaller chunks for storage
• Allows you to make statements about a resource from a different location
Exercise

• What graph do these two files create?
  – http://modusoperandi.com/test1/:
    @prefix mo: <http://modusoperandi.com/ont#> .
    mo:JudyG mo:hasPhone "321-555-1212" .
    mo:JudyG mo:authorOf
  
  – http://modusoperandi.com/test2/:
    @prefix mo: <http://modusoperandi.com/ont#> .
    mo:JudyG mo:worksFor mo:ModusOperandi .
    mo:ModusOperandi mo:hasWebSite
Blank Nodes

- RDF Graph Model allows "blank" nodes (No URI)
- Cannot be referenced from "outside" the file or triple store it is defined in
- Serialized different ways:
  - [ ] (square braces) in Turtle
    - e.g. [ ] a foaf:Person ; foaf:name "Judi Gasperini"
  - no resource value in XML
    - e.g.
      - <foaf:Person>
      -  <foaf:name>Judi Gasperini</foaf:name>
      -  </foaf:Person>
  - "_:" prefix in NTriples
    - e.g. _:b1 or _:A476f88c8X3aX137b2995e69X3aXX2dX7ff8
    - _:b0 <http://xmlns.com/foaf/0.1/name> "Judi Gasperini"
How is RDF "Semantic?"

• RDF is "semantic" in that, through globally unique resource names (URIs), everyone "agrees" on the meaning of terms

• E.g. foaf:knows (i.e., <http://xmlns.com/foaf/0.1/knows>) means "A person known by this person" the way the FOAF standard defines it
  – If you use foaf:knows in your document, and someone else uses it in theirs, you mean the same thing

• No inference with basic RDF
  – i.e., the triples you state are the only triples you get
RDF: What We Learned

• RDF is a Graph Model
• Graph can be represented using Triples
• Resources vs. Literals
• Serialization (N-Triples, RDF/XML, Turtle)
• Querying = graph matching
• RDF graph can be separated over many files/stores
• Blank Nodes are allowed
• Semantics come from agreement on vocabulary
What is RDFS?

• RDFS = RDF Schema
• A type system for RDF
• Supports Inference!
• W3C Recommendation (2004)
RDF vs. RDFS

• RDF ...
  – provides way to express simple statements about resources

• RDF...
  – provides no means for defining application-specific classes (kinds of resources) and properties (specific properties describing resources)
    • Class examples: Person
    • Property examples: name, knows

• RDFS...
  – Provides a type system for RDF
RDFS Type System

• Similar to OO in some ways:
  – Resources can be defined as instances of one or more classes
  – Classes can be organized in hierarchical fashion

• Different in some ways:
  – Class and property descriptions do not restrict information, but provide additional information about the resources they describe
  – Properties exist independently from classes!
  – Properties can be organized in hierarchical fashion
  – Multiple inheritance is allowed/encouraged!
RDFS is RDF

• Prefix
  – http://www.w3.org/2000/01/rdf-schema#
  – Conventionally, uses QName prefix "rdfs:"
    • E.g. rdfs:Class, rdfs:subClassOf

• Key constructs it defines:
  – rdf:type
  – rdfs:Class
  – rdf:Property
  – rdfs:subClassOf
  – rdfs:subPropertyOf
  – rdfs:domain
  – rdfs:range
  – rdfs:label
  – rdfs:comment
RDFS Classes

• A Class is a set of things
• An RDF resource represents a Class if it has a `<rdf:type>` property whose value is `<rdfs:Class>`

  – As triple: `mo:FemalePerson <rdf:type> <rdfs:Class>`.

• A resource can be a member of that class (set)

  – As triple: `mo:JudyG <rdf:type> mo:FemalePerson`.

©2012 Semantic Technology Conference
June 3-7, 2012
RDFS Subclasses

• Specialization can be described using rdfs:subClassOf
  – E.g.: mo:FemalePerson rdf:type rdfs:Class .
    mo:FemalePerson rdfs:subClassOf foaf:Person .
    mo:Mother rdf:type rdfs:Class .
    mo:Mother rdfs:subClassOf mo:FemalePerson .

• rdfs:subClassOf is transitive
  – E.g.,
    if every mother is a female person,
    and every female person is a person,
    then every mother is also a person
RDFS Classes and Subclasses

• How to Assert
  – A \texttt{rdfs:subClassOf} B .
  – E.g.
  – \texttt{mo:FemalePerson rdfs:subClassOf foaf:Person} .

• Meaning
  – If you are of type A, you are also of type B
  – If you are in set A, you are in set B

• Inference
  – Assert:
    • \texttt{mo:FemalePerson rdfs:subClassOf foaf:Person} .
    • \texttt{mo:JudyG a mo:FemalePerson} .
  – Infer:
    • \texttt{mo:JudyG a foaf:Person} .
Exercise

• Arrange the following Classes into a hierarchy using RDFS
  – Person
  – Female Person
  – Male Person
  – Animal
  – Mammal
So what is inference?

- Creating new triples from existing triples, or...
- Deriving information that was not in the original data, or...
- Adding facts
- E.g.
  - RDF:
    - Assertion: `mo:MarkW rdf:type mo:MalePerson`
    - we get nothing else
  - RDFS:
    - Assertion: `mo:MarkW rdf:type mo:MalePerson`
    - Assertion: `mo:MalePerson rdfs:subClassOf foaf:Person`
    - Inference (new triple): `mo:MarkW rdf:type foaf:Person`
    - We added a fact not in the original data
RDFS Properties and Subproperties

• How to Assert
  – B \texttt{rdfs:subPropertyOf} A

• Meaning
  – if \( x \texttt{B} y \), then \( x \texttt{A} y \)
  – e.g.: \( \texttt{mo:brotherOf rdfs:subPropertyOf mo:siblingOf} \)
  – implies: if \( X \texttt{mo:brotherOf} Y \), then \( X \texttt{mo:siblingOf} Y \)

• Inference
  – Assert:
    • \( \texttt{mo:brotherOf rdfs:subPropertyOf mo:siblingOf} \)
    • \( \texttt{mo:MarkW mo:brotherOf mo:JohnW} \)
  – Infer:
    • \( \texttt{mo:MarkW mo:siblingOf mo:JohnW} \) (a new triple)

• \texttt{rdfs:subPropertyOf} is \textit{transitive}
  – i.e., if A \texttt{rdfs:subPropertyOf} B, and B \texttt{rdfs:subPropertyOf} C, then A \texttt{rdfs:subPropertyOf} C
Exercise

• What can we infer from these RDFS statements:

<table>
<thead>
<tr>
<th>Property</th>
<th>RDFS SubPropertyOf</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>mo:brotherOf</td>
<td>rdfs:subPropertyOf</td>
<td>mo:siblingOf</td>
</tr>
<tr>
<td>mo:sisterOf</td>
<td>rdfs:subPropertyOf</td>
<td>mo:siblingOf</td>
</tr>
<tr>
<td>mo:siblingOf</td>
<td>rdfs:subPropertyOf</td>
<td>mo:relativeOf</td>
</tr>
<tr>
<td>mo:Mark</td>
<td>mo:brotherOf</td>
<td>mo:John</td>
</tr>
<tr>
<td>mo:Mark</td>
<td>mo:siblingOf</td>
<td>mo:Barb</td>
</tr>
<tr>
<td>mo:Barb</td>
<td>mo:sisterOf</td>
<td>mo:John</td>
</tr>
</tbody>
</table>
RDFS "Individuals"

• Any resource with `rdf:type` of `rdfs:Class` is a Class
• Any resource with `rdf:type` of `rdf:Property` is a Property
• Any other use of `rdf:type` denotes an individual (instance) belonging to a class
• E.g.
  – FemalePerson  `rdf:type` `rdfs:Class`
  – JudyG  `rdf:type` FemalePerson
  – MarkW  `rdf:type` Person
    • (implies Person is a class)
  – ECO2015  `rdf:type` CollegeCourse
    • (implies CollegeCourse is a class)
RDFS Domain

• How to Assert
  – P rdfs:domain C (where P is a property, C a Class)
  – e.g., mo:motherOf rdfs:domain mo:FemalePerson.

• Meaning
  – the subjects of triples using property P are of type C
  – e.g., the subjects of mo:motherOf statements must be of type mo:FemalePerson

• Inference
  – Assert:
    • mo:motherOf rdfs:domain mo:FemalePerson.
    • mo:Trilby mo:motherOf mo:MarkW.
  – Infer:
    • mo:Trilby rdf:type mo:FemalePerson.
RDFS Range

• How to Assert
  – P rdfs:range C (where P is a property, C a Class)
  – e.g., mo:motherOf rdfs:range foaf:Person .

• Meaning
  – the objects of triples using property P are of type C
  – e.g., the objects of mo:motherOf statements must be of type foaf:Person

• Inference
  – Assert:
    • mo:motherOf rdfs:range foaf:Person .
    • mo:Trilby mo:motherOf mo:MarkW .
  – Infer:
    • mo:MarkW rdf:type foaf:Person .

©2012 Semantic Technology Conference
June 3-7, 2012
RDFS Domain and Range

• Often called domain and range "restrictions"
• But, inference will enforce them by **adding** triples (not reporting an error)
  – E.g. We assert:
    • mo:Necessity rdf:type mo:NounWord .
    • mo:motherOf rdfs:domain mo:FemalePerson .
    • …
    • mo:Necessity mo:motherOf mo:Invention .
  – RDFS reasoner will add:
    • mo:Necessity rdf:type mo:FemalePerson

• So, yes it is a "restriction": all domains of the property must be a Female Person, but... the way this plays out is to "make it so", not report an error
• Domain and Range can be left unstated
• OK to have multiple domains and ranges per property
Exercise

• What triples can we infer from these statements
  – FemalePerson rdfs:subClassOf Person
  – motherOf a rdf:Property
  – motherOf rdfs:domain FemalePerson
  – motherOf rdfs:range Person
  – Trilby motherOf MarkW
RDFS Labels and Comments

• Useful for annotating resources
• rdfs:label
  – short text description
  – e.g.

```xml
<http://books.com/MobyDick>
  rdfs:label  "Moby Dick"@en , "Historias de Siempre"@sp .
```

• rdfs:comment
  – longer text description, e.g.

```xml
<http://books.com/MobyDick> rdfs:comment
  """"A rich, complex, highly symbolic narrative that explores the deepest reaches of our moral and metaphysical dilemma through the extraordinary tale of Captain Ahab's insane quest for the great white whale. One of America's greatest novels. """"@en .
```
How Inference Affects RDF Queries

- Remember: RDF queries simply match a graph
- If a triple is not in the graph, it will not be matched
- E.g. without running a reasoner, if I only assert:
  mo:JudyG a mo:FemalePerson
  mo:FemalePerson rdfs:subClassOf foaf:Person
- and then query
  select * where {?p a foaf:Person}
- I will get no results!
  mo:JudyG a foaf:Person is not in the graph I asserted
- Remember: **Inference adds triples** to a graph
  – (You must) use a reasoner to add the facts implied by your statements
Best Practices

• Domain/range are not restrictions, but rather add triples; be careful how you use them

• Provide label and comment for every concept
  – Comment: include an example
  – E.g.

    HostDevice a rdfs:Class ;
    rdfs:label "Host Device"@en ;
    rdfs:comment """"A host or computer,
    e.g. Host_1 a HostDevice .
    Host_1 hasIpAddr "192.168.0.1" . """"@en.
How is RDFS "Semantic?"

• It has all the semantic aspects of RDF
• It adds constructs with specific meanings for inference, thus reasoners can add facts based on the ones asserted
• However, you do not get "validation" with RDFS
  – All statements are "true"
  – No contradictions possible in RDFS
What Can You Do with RDFS?

• Anything you can do with RDF
• Relate different RDF Schemas using:
  – Sub-class statements
  – Sub-property statements
• Infer information:
  – Classification: given what you know about a resource, you can determine what classes it belongs to (domain, range, subClassOf)
  – Add connections to the graph through property expansions (subPropertyOf)
RDFS: What We Learned

• RDFS provides a type system for RDF
• Classes & Properties
• SubClassOf & subPropertyOf
• Domain & Range
• Inference = adding triples to graph that were not explicitly stated (via a reasoner)
• RDFS good for relating different schemas
• Semantics of RDF + inferring information not stated
What is OWL?

• OWL = Web Ontology Language
• Extends richness of RDFS to the level of ontologies, for richer inference
• Supports Validation (Consistency Checking)
• W3C Recommendation (2004, 2009)
• See http://www.w3.org/TR/owl2-overview/
OWL is RDF

• Prefix
  – http://www.w3.org/2002/07/owl#
  – Conventionally, uses QName prefix "owl:'
  – E.g. owl:Class, owl:inverseOf

• Builds upon RDFS

• In some ways restricts RDFS

• Syntactically, OWL, RDFS, and RDF are the same

• The difference is the inference
OWL adds to RDFS

Equality/Inequality:
equivalentClass
disjointWith
equivalentProperty
sameAs; differentFrom
AllDifferent
distinctMembers

Property Characteristics:
ObjectProperty
DatatypeProperty
inverseOf
TransitiveProperty
SymmetricProperty
FunctionalProperty
InverseFunctionalProperty

Restricted Cardinality:
minCardinality
maxCardinality
cardinality

Property Restrictions:
Restriction
onProperty
allValuesFrom
someValuesFrom
hasValue

Class Definition:
owl:Class
intersectionOf
unionOf
complimentOf
oneOf

Header Information:
Ontology
imports

Annotation Properties:
AnnotationProperty
OntologyProperty

Versioning:
versionInfo
priorVersion
backwardCompatibleWith
incompatibleWith
DeprecatedClass
DeprecatedProperty

AND MORE!
keys;
property chains;
richer datatypes, data ranges;
qualified cardinality restrictions;
asymmetric, reflexive, and disjoint properties; and enhanced annotation capabilities
More restricted than RDF and RDFS

- A property must be declared as either Object or Datatype property
- Except for OWL-Full, a Resource must be used only as one of: Class, Property, Individual
- Uses RDFS to extend RDF
OWL Individuals

• As in RDFS, represent instances of a class
• OWL adds `owl:Class` as a subclass of `rdfs:Class`
• An individual if: subject of an `rdf:type` statement where range is an `owl:Class`, e.g.
  – Person `rdf:type` `owl:Class`
  – JudyG `rdf:type` Person <= declares an individual
owl:ObjectProperty, owl:DatatypeProperty

- **owl:ObjectProperty** - property will only have ranges that are Resources (not-literals)
  - Links two individuals together
  - E.g.: foaf:knows rdf:type owl:ObjectProperty
  - MarkW foaf:knows JudyG

- **owl:DatatypeProperty** - property will only have ranges that are literals
  - Links individual to a value
  - E.g.: foaf:name rdf:type owl:DatatypeProperty
  - JudyG foaf:name "Judy Gasperini"

- This distinction required for Description Logics (DL) reasoning
owl:sameAs, owl:differentFrom

• On the web, two different URIs could represent the same thing
• The same is true in RDF/OWL: different Resource URIs could be same thing – can't tell
• A.k.a. "No unique naming assumption"
• owl:sameAs – assert two Resources are the same
• owl:differentFrom – assert two Resources are different
• E.g.
  – NewYorkCity owl:sameAs TheBigApple
  – NewYorkCity owl:differentFrom NewOrleans
owl:inverseOf

• Declares a property to be the inverse of another
• Useful for merging ontologies
• E.g.
  – Assert:
    – mo:hasChild owl:inverseOf mo:hasParent
  – mo:MarkW mo:hasChild mo:DavidW
  – Infers:
    – mo:DavidW mo:hasParent mo:MarkW
owl:TransitiveProperty

- If A related to B, and B to C by same relation, then A related to C by that same relation
- Often used for part-of or ancestor relationships
- E.g.
  - Assert:
    - `mo:hasAncestor rdf:type owl:TransitiveProperty`
    - `mo:MarkW mo:hasAncestor mo:LewWJr`
    - `mo:LewWJr mo:hasAncestor mo:LewWSr`
  - Infer:
    - `mo:MarkW mo:hasAncestor mo:LewWSr`
owl:SymmetricProperty

• If A related to B by property, B is related to A by the same property

• E.g.
  – Assert:
    – mo:hasSibling rdf:type owl:SymmetricProperty
    – mo:MarkW mo:hasSibling mo:JohnW
  – Infer:
    – mo:JohnW mo:hasSibling mo:MarkW
owl:FunctionalProperty

• if P is functional, then individual A can have property P to only one object

• E.g.
  hasBiologicalMother rdf:type
  owl:ObjectProperty, owl:FunctionalProperty .
  hasSSN a
  owl:DatatypeProperty, owl:FunctionalProperty .

• How is this useful?
  – To infer consistency conflicts
  – To infer two things are the same
Exercise

• What will reasoner say based on these facts?
  – hasBiologicalMother rdf:type owl:FunctionalProperty
  – Mark hasBiologicalMother Mom1
  – Mark hasBiologicalMother Mom2

• What will reasoner say based on these facts?
  – hasBiologicalMother rdf:type owl:FunctionalProperty
  – Mark hasBiologicalMother Mom1
  – Mark hasBiologicalMother Mom2
  – Mom1 owl:differentFrom Mom2
• if P is inverse functional, then individual A can have property P from only one subject

• E.g. 
  isBiologicalMotherOf rdf:type owl:ObjectProperty, 
  owl:InverseFunctionalProperty .

• How is this useful?
  – To infer two things are the same
    • (Note: foaf:email is an InverseFunctionalProperty)
  – To infer consistency conflicts
owl:equivalentClass

- Like sameAs but for classes
- Equivalent classes have same members
- Good for merging graphs / ontologies
- E.g.
  - foaf:Person  **owl:equivalentClass**  mo:Person
owl:oneOf

- Defines a class by listing the only individuals in it
- I.e. an "enumerated class"
- E.g.

```owl
mo:WineColor a owl:Class ;
owl:equivalentClass [
 a owl:Class ;
 owl:oneOf (mo:Red mo:Rose mo:White)
].

[] a owl:AllDifferent ;
owl:distinctMembers (mo:Red mo:Rose mo:White).
```

- Usually used with owl:differentFrom or owl:AllDifferent
owl:disjointWith

- All classes could potentially overlap
- owl:disjointWith asserts that there is no overlap (no members in common)
- E.g.
  - FemalePerson a Person.
  - MalePerson a Person.
  - FemalePerson owl:disjointWith MalePerson.
- Is symmetric
- How is this useful?
  - To infer consistency conflicts (if an individual is determined to be member of two disjoint classes)
  - Often in class hierarchies, subclasses at the same level are disjoint
Class Descriptions

• Classes can be described by their logical characteristics
  – E.g.
    • union of 2 or more classes
    • intersection of 2 or more classes
    • complement of a class
    • by what properties their members have

• Restrictions can be combined
  – "union of the intersection of ..."

• Anonymous definitions are often used
  – E.g.
    – Class A subclass of (intersection of classes B and C)
Local Property Range Restrictions

- RDFS gave us "global" range restrictions
- OWL supports range restrictions local to a class
- E.g. owl:allValuesFrom
  - for individuals of class "Human", all ranges of "hasChild" property are also "Human"

```xml
<owl:Class rdf:about="#Human">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:allValuesFrom rdf:resource="#Human"/>;
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

things where all hasChild properties point to Humans

Human
Types of Local Restrictions

• More restrictions that can be placed on particular properties for a class:
  – someValuesFrom - at least one property value comes from Class X
    • e.g. Parent class has at least one hasChild property value from class Person
  – hasValue - property P has specific value V
    • e.g. for Red Wine class, wineColor property has value "Red"
  – minCardinality - minimum of N instances of property P
  – maxCardinality - all class members have maximum of N instances of property P
  – cardinality (exact) - all class members have exactly N instances of property P
Open-World Assumption

- A.k.a. "OWA"
- Opposite of Closed-World Assumption
  - "negation as failure"
  - i.e. if "Mark" not in Customer DB, then Mark is not a customer
- OWA says: "A triple may be out there that we don't know about yet"
  - Think internet... someone may have asserted something in some doc we haven't read yet
- E.g.
  - If I assert: class ParentOfOnlySons has all values of hasChild from MalePerson
  - And I assert:
    - Mark hasChild David
    - David a MalePerson
  - I cannot conclude that Mark is in class ParentOfOnlySons
    - (Because Mark may have another hasChild property somewhere in the open world that links to an individual who is not a MalePerson)
TBox and ABox

• TBox = concept definitions (terms)
  – I.e. Class and property definitions
  – E.g.
    foaf:Person a owl:Class .
    mo:FemalePerson a owl:Class .
    mo:FemalePerson rdfs:subClassOf foaf:Person.
    foaf:knows a owl:ObjectProperty .

• ABox = individual definitions (assertions)
  – I.e. Individuals and their properties
  – E.g.
    mo:JudyG a mo:FemalePerson .
    mo:JudyG foaf:knows mo:MarkW .
    mo:JudyG mo:hasPhone "321-555-1212" .
Importing Ontologies

• Ontologies can import other ontologies
• Allows modularization
• Allows reuse of third-party ontologies

```turtle
<http://modusoperandi.com/ont>
    rdf:type owl:Ontology ;
```

```owl
<owl:Ontology rdf:about="http://modusoperandi.com/ont">
    <owl:imports rdf:resource="http://xmlns.com/foaf/0.1"/>
</owl:Ontology>
```
OWL Dialects

• OWL 1 (2004) defined two major dialects:
  – OWL DL and OWL Full
  – and one syntactic subset: OWL Lite

• These proved insufficient

• OWL 2 (2009) defines three different profiles:
  – OWL 2 EL, OWL 2 QL, and OWL 2 RL
  – sublanguages (syntactic subsets) of OWL 2 with useful computational properties or implementation possibilities
    • EL - large-scale ontologies (many classes and property definitions)
    • QL - sound and complete query answering, while having tight integration with RDBMSs
    • RL - scalable reasoning while amenable to implementation using rule-based technologies

  – See http://www.w3.org/TR/owl2-profiles/
OWL: What We Learned

• Individuals
• owl:ObjectProperty & owl:DatatypeProperties
• owl:sameAs, owl:differentFrom
• owl:inverseOf
• owl:TransitiveProperty
• owl:SymmetricProperty
• owl:FunctionalProperty, owl:InverseFunctionalProperty
• owl:equivalentClass, owl:oneOf, owl:disjointWith
• Union, intersection, complement of classes
• Local Property Range Restrictions
• Open World Assumption
• TBox vs. ABox
• Importing ontologies from other ontologies
• There are multiple OWL dialects
That's all, Folks!

– Thanks for coming!

– Blog: http://semapps.blogspot.com

– Contact me: mwallace@modusoperandi.com