Semantic Web

Lectures 7 and 8: RDFS & OWL

Knarig Arabshian
Knarig.arabshian@hofstra.edu

How can a model help us?

- Models help people communicate
  - Describe situation in a particular way that other people can understand
- Models describe and predict
  - Relates primitive concepts to one another and to more complex ones
- Models common ground for many viewpoints
  - Two people may not agree on how they perceive a domain but model can be a medium between two viewpoints

Modeling for communication

- Human to human communication tends to use informal modeling
- Informal modeling on the web
  - E-commerce systems
    - Online catalogs
    - Most online catalogs use similar modeling techniques
  - Tagging systems
    - community tagging
    - If any two people use the same tag, this becomes a common organizing entity
    - Tagging infrastructure shows which tags are used most
    - Helps browsers determine what tags to use in a search

Modeling for knowledge and prediction

- Models can be re-used to explain or predict information
- Example:
  - Knowledge
    - Understanding how fire is put out can help you understand why using a blanket helps put out a fire
  - Prediction
    - Know: ice is slippery
    - Predict:
      - Holding an ice cube might cause it to slip through your fingers
      - Driving on ice might cause your car to skid
      - Effective skating on ice would require blades

Modeling for knowledge and prediction

- This requires formal modeling
- Formal models are basic component of mathematics
- Basic arithmetic:
  - Agree on the numbering system (what does the symbol '1', '2', and '3' mean?)
  - Agree on symbols (what does '+' or '=' mean?)
  - Indisputable fact: 1 + 2 = 3

Models Help Establish Common Grounds

- Variability is something to be expected
  - Example: is pluto a planet? Astrologers think it is; astronomers now consider it a dwarf planet.
- How to handle this?
  - Adopt the preferred point of view
  - Represent both points of view and let the information consumer decide how to model the information given the two differing models
    - This is current state of the Web

Modeling in Semantic Web

- Create an infrastructure that allows:
  - Any topic to be discussed
  - Community to work through misunderstandings
  - A framework to represent and organize common or different viewpoints (taxonomy)
- Layers of expressivity
  - Expressing the concept of “water”
    - Molecular structure (H2O, H--O--H)
    - Different states (liquid, solid, vapor)

Inferencing in Semantic Web

- Relationships need to be added to the data so that it will seem more connected and consistently integrated
- Want to express the relationship between “apple” and “fruit”
- Referring to apple, also refers to fruit
- Restaurant in a specific location also means that the location contains this restaurant

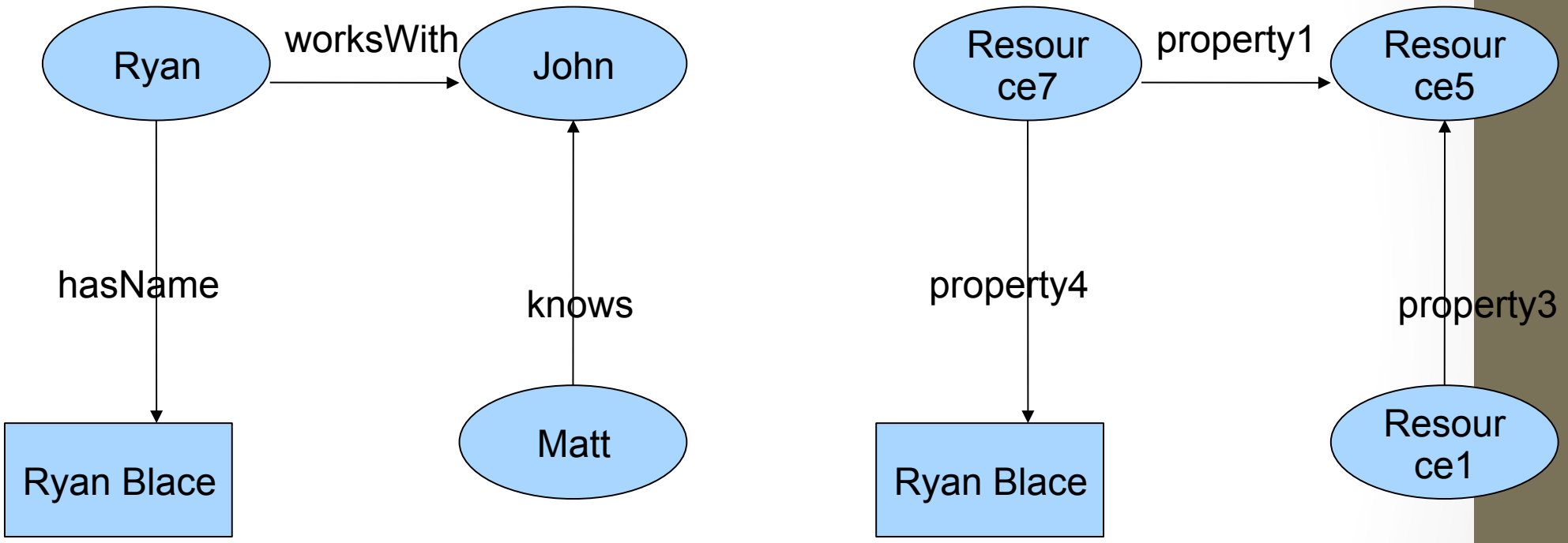
Inferencing in the Semantic Web

- RDF allows us to express propositions for single resources:
  - “Book hasName FOSWT”
  - “Book hasAuthor Hitzler”
- RDF describes *individuals* or *instances* such as “Book”, “Hitzler”, “FSWT” and puts them in relation to one-another

Inferencing in Semantic Web

- RDF lets individuals have limited types
  - Types indicate the class of entities an individual or instance belongs to
  - Literals can be in the class of natural numbers or character strings or ordered lists
- Need more information to understand the semantic meaning of a resource
  - What class of entities are these part of? Can't express this
    - “Pascal Hitzler” (person)
    - FOSWT (title)
    - Book (printed media)

Inferencing in Semantic Web



- RDF provides a way to model information
- Does not provide a way of specifying what that information means
- RDF graph alone can only be interpreted as a graph
- Meaning is apparent on your ability to recognize and interpret the URIs, literals, and general structure of graph

Inferencing in Semantic Web

- Vocabulary
  - Terms referring to individuals, relations and classes
  - User will have concrete idea about meanings
    - Restaurant is an Eatery
    - University is an Institution
    - Pascal Hitzler is a Person
  - To the computer system the vocabulary is just a series of character strings with no meaning
  - Semantic interrelations need to be explicitly communicated to the system in some format in order to enable it to draw conclusions that rely on human knowledge

RDF Review

- RDF has three basic kinds of descriptive elements:
  - *Individuals* (authors of a textbook, publisher or a cooking recipe)
  - Put into *relation* to each other
  - Possible to assign *types* to literals and resources and lets you state that they belong to a class of entities that share certain characteristics (natural numbers or ordered lists)
- When describing new domains of interest, new terms need to be introduced for individuals (“Knarig Arabshian” or “HofstraUniversity”) and relations (employed by)
- Humans understand university is an institution or persons can be employed by an institution
- But computer systems need to be told this via a ‘schema’.

RDF Schema (RDFS)

- Vocabulary knowledge is specified in a schema knowledge
- Uses RDF vocabulary
  - Well-formed RDF document
  - Read and processed by all tools that support RDF
  - But those parts that are specifically defined for RDFS (semantics) will not be understood but can be parsed
- URI: http://www.w3.org/2000/01/rdf-schema# abbreviated by rdfs:

RDFS

- RDFS is a knowledge representation language or ontology language
  - Provides means for describing some types of semantic interdependencies of a domain of interest
- Provides generic language constructs
  - User-defined vocabulary which characterizes semantics
  - Example: subClassOf → property that indicates that one thing is a subclass of another

RDFS

- RDFS provides a machine-processable ontology language of a domain of interest
- Formally defined meaning given by formal semantics of RDFS
- Useful as a simple lightweight ontology language but has limitations
- OWL (Web Ontology Language) is more expressive but at the expense of speed
  - Runtime for automated inference is inversely proportional to expression

Classes and Instances

- Classes contain a set of resources (think of sets in set theory)
- Knowledge specification formalism needs to provide a way to "type" resources
- URI rdf:type used to make resources as instances of a class (belongs to a class)
- Separate semantics and syntax
  - Term "class" means set of resources
  - Term "class names" are URIs that represent or refer to a class

Class and Instances

- book:uri rdf:type ex:Textbook .
  - Describe this FOSWT as a textbook.
    - Semantically this means that FOSWT is a member of the class of all textbooks
- A single URI can't distinguish whether or not book:uri or ex:Texbook is a class or instance
- Need to model this formally within an RDFS document
- RDFS provides a way to indicate class names by “typing” them as classes
- So ex:Textbook can be specified as a class by typing it as a class

Classes and Instances

- ex:Textbook rdf:type rdfs:Class .
  - URI ex:Textbook is typed as a class name
- book:uri rdf:type ex:Textbook .
  - This now means that ex:Textbook is a class and instances within this class will be textbooks
  - book:uri is a resource that's an instance of the textbook class
- rdfs:Class rdf:type rdfs:Class .
  - Note: the class of all classes is also a class and contained in itself

Classes in RDFS

- rdfs:Resource: class of all resources
- rdfs:Property
  - Class of all properties
  - All resources that stand for relations
- rdfs:Literal
  - Class of all literal values, implies that it comprises all datatypes as subclasses
- rdfs:XMLLiteral
  - Class of all values that are literals
- rdfs:Datatype
  - Class of all datatypes (example, XML literal)
- rdf:Bag, rdf:Alt, rdf:Seq, rdfs:Container
  - Class of lists

Classes in RDFS

- URIs representing classes are capitalized
- Instances are written in lower case
- Classes are not necessarily nouns
  - Can be adjectives like ex:Organic or ex:Red
- Class membership is not exclusive
  - Resource can belong to several different classes

```
book:uri rdf:type ex:Textbook .
book:uri rdf:type ex:WorthReading .
```

Subclasses and Class Hierarchies

- Simply defining classes is not enough

book:uri rdf:type ex:Textbook .

- When looking for FOSWT book in the class of books ex:Book, it will not be found because it has been instantiated as a ex:Textbook
- But every textbook is also a book and every instance of the class ex:Textbook should also be an instance of the class ex:Book

Subclasses and Hierarchies

- One solution:
  - Add another triple book:uri rdf:type ex:Book .
  - Same problem would occur if another resource typed as a textbook is instantiated
  - Therefore, for any triple of type ex:Textbook, another triple of type ex:Book would have to be instantiated

    u rdf:type ex:Textbook .

    u rdf:type ex:Book .
  - These steps would have to be repeated every time new information is specified
  - Not a good solution

Subclasses and Hierarchies

- Solution
  - Specify that every textbook is also a book
  - Class of all textbooks is comprised by the class of all books
  - Textbook is a subclass of book
  - Book is a superclass of textbook
  - Use the predicate rdfs:subClassOf to specify this relationship

ex:Textbook rdfs:subClassOf ex:Book .

  - Now, book:uri will also be regarded as a book even if it is not explicitly stated

Subclasses and Hierarchies

- Class hierarchies can be modeled with this structure
  - Indicate that book is also a type of printed media
  - Journal is also a type of printed media

ex:Book rdfs:subClassOf ex:PrintMedia .

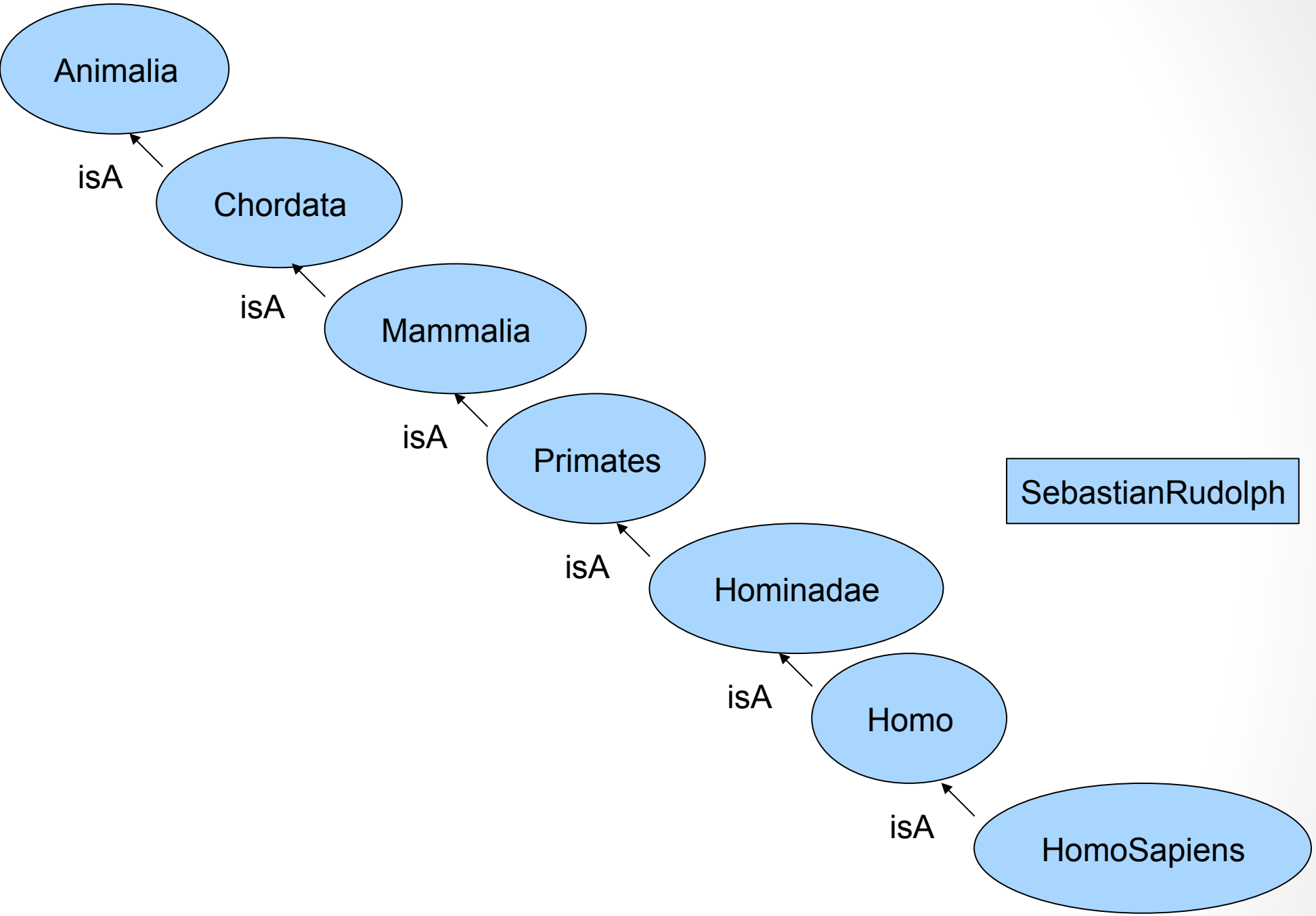
ex:Journal rdfs:subClassOf ex:PrintMedia .

- Subclass relationships are transitive: subclasses of subclasses are subclasses
  - Following triple is deduced from top two triples

ex:Textbook rdfs:subClassOf ex:PrintMedia .

- Subclass relationships are reflexive: class is its own subclass

ex:Book rdfs:subClassOf ex:Book .


Animalia
isA
Chordata
isA
Mammalia
isA
Primates
isA
Hominadae
isA
Homo
isA
HomoSapiens
SebastianRudolph

RDFS Example

- http://www.cs.columbi.edu/~knarig/animalia.rdfs

Example: Write out an RDF Schema of the following B&N book categorization

Home › Books › Bestsellers › New York Times Bestsellers › Paperback Trade Fiction

The New York Times® Bestsellers

For the Week of February 13, 2011

Bestselling Books

- This Hour's Top 100
- Top 100 NOOKbooks
- B&N Weekly Store Bestsellers
- Bestsellers by Subject
- Paperback Bestsellers
- The Year's Top 100
- The New York Times Bestsellers

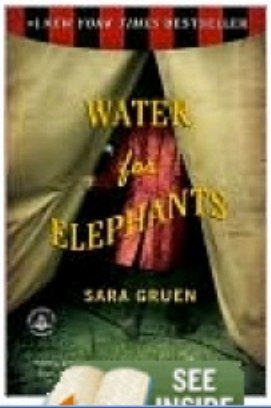
The New York Times Bestsellers

- Hardcover Fiction
- Trade Paperback Fiction

Trade Paperback Fiction

Browse Fiction Bestsellers:
Harcover | Paperback | NOOKbook | Mass Market

The New York Times® is the registered trademark of The New York Times® Company, which is not affiliated with and does not sponsor or endorse the services of BN.com.

1. 

Water for Elephants
by Sara Gruen
(Paperback)

Usually ships within 24 hours

List Price: $14.95
Online Price: **$8.17**
(You Save 45%)

Add to Cart

Add To List

Example

ex:Book	rdf:type	rdfs:Class
ex:BestSellers	rdfs:subClassOf	ex:Book
ex:NYTBestSellers	rdfs:subClassOf	ex:BestSellers
ex:PaperBackFiction	rdfs:subClassOf	ex:NYTBestSellers
ex:WaterForElephants	rdf:type	PaperBackFiction

RDFS Properties

- Labeled by URIs and considered a special kind of resource
- Not a class or individual (instance)
- Describe relations between “proper” resources or individuals (meaning subject and object)
- Mathematics: relation is represented as a set of the pairs interlinked by that relation
- ex:isMarriedTo would be the set of all married couples
- So properties can resemble classes more than individuals

RDFS Properties

- RDF vocabulary provides the class name rdf:Property
  - Class of all properties
  - Expresses that a URI refers to a property
- Now, the ex:publishedBy property can be assigned to the property class by assigning the corresponding type

  ex:publishedBy rdf:type rdf:Property .
- rdf:Property denotes a class and not a property. It contains properties as instances.
- In addition to explicitly typing a URI as a property, if a URI is used as a property within a triple, RDFS semantics ensures that this is considered as a property

book:uri ex:publishedBy crc:uri .

RDFS Subproperties and Property Hierarchies

- RDFS allows the specification of subproperties
  - ex:isHappilyMarriedTo property can be specified as a subproperty of ex:isMarriedTo property
  - Expressed in RDFS as follows:

  ex:isHappilyMarriedTo rdf:subPropertyOf ex:isMarriedTo .
- Can deduce from the following triple:

  ex:markus ex:isHappilyMarriedTo ex:anja.

  Also means that

  ex:markus ex:isMarriedTo ex:anja
- One subproperty statement is needed to enable RDFS compliant information system to automatically recognize all pairs recorded as "happily married" and "married"

Property Restrictions

- When two entities are connected by a certain property, it lets us draw conclusions about how these classes relate to one another.
- One such relationship will be class membership.
- Example: statement that one entity is married to another implies that both involved entities are persons
- Whenever following triple occurs:

```
a  ex:isMarriedTo      b  .
```

want to assert that following triples are valid:

```
a   rdf:type      ex:Person .
b   rdf:type      ex:Person .
```

Property Restrictions

- Explicitly adding these class membership statements to RDF document is cumbersome and repetitive
- Good to have a “macro” or “template” like mechanism which is entered once and ensures that these class memberships are imposed by the properties
- Solution: specify the domain and range of classes a property belongs to
- Domain will classify the subjects
- Range will classify the objects

Property Restrictions

- RDFS vocabulary provides following two properties which allow this to happen
  - rdfs:domain specifies a predicate relating a property to a domain class – classifies subjects
  - rdfs:range specifies a predicate relating a property to a range class – classifies objects

```
ex:isMarriedTo    rdfs:domain    ex:Person  .
ex:isMarriedTo    rdfs:range  ex:Person  .
```

- Literal values can also be classified as an object of a certain property

```
ex:hasAge rdfs:range  xsd:nonNegativeInteger   .
```

Property Restrictions

- Domain and range restrictions constitute the “semantic link” between classes and properties
- Properties link classes together and relate one to another
- Domain and range specifications allow you to specify the type of classes that are linked to one another given a certain property name

Property Restrictions

ex:authorOf rdfs:range ex:Textbook .

ex:authorOf rdfs:range ex:Storybook .

- According to RDFS semantics this expresses that every resource in the range of an authorship relation is BOTH a textbook and a storybook
- Does not mean that someone may be author of a textbook or a storybook
- Every declared property restriction globally affects all occurrences of this property
- Need to be careful when restricting properties
- Make sure that there is always a sufficiently general class that contains all possible resources that might occur in the subject

Property Restrictions

```
ex:isMarriedto      rdfs:domain      ex:Person   .
ex:isMarriedTo      rdfs:range       ex:Person   .
ex:instituteAIFB    rdf:type         ex:Institution   .
ex:pascal           ex:isMarriedTo   ex:instituteAIB .
```

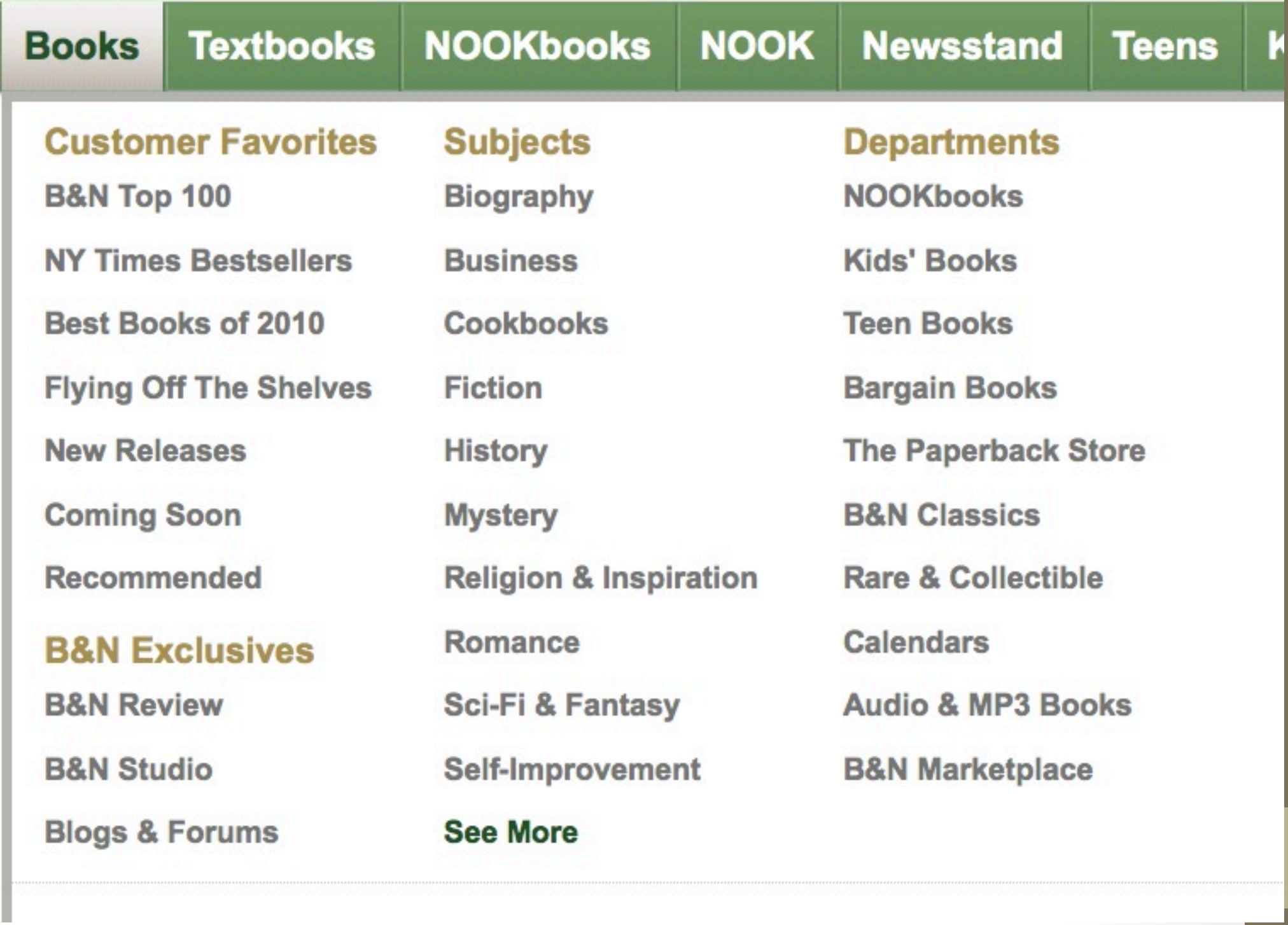
Type mismatch problem is showing up here

- We would expect that this statement is automatically rejected
- But RDF range and domain statements do not carry constraint semantics thus consequence of above triples becomes:

```
ex:instituteAIFB    rdf:type         ex:Person   .
```

Examples

- B&N cont'd
  - Categorize books
  - Use classification on the B&N site
  - Create class hierarchy
  - Create properties and specify their domain and ranges
- Example from FOWST


Books
Textbooks
NOOKbooks
NOOK
Newsstand
Teens
Customer Favorites
B&N Top 100
NY Times Bestsellers
Best Books of 2010
Flying Off The Shelves
New Releases
Coming Soon
Recommended
B&N Exclusives
B&N Review
B&N Studio
Blogs & Forums
Subjects
Biography
Business
Cookbooks
Fiction
History
Mystery
Religion & Inspiration
Romance
Sci-Fi & Fantasy
Self-Improvement
See More
Departments
NOOKbooks
Kids' Books
Teen Books
Bargain Books
The Paperback Store
B&N Classics
Rare & Collectible
Calendars
Audio & MP3 Books
B&N Marketplace

Example

```
ex:Book                    rdf:type            rdfs:Class          .
ex:CustomerFavorite        rdfs:subClassOf     ex:Book             .
ex:Subject                 rdfs:subClassOf     ex:Book             .
ex:BNExclusive             rdfs:subClassOf     ex:Book             .
ex:Department              rdfs:subClassOf     ex:Book             .

ex:hasCustomerFavorite     rdf:type    rdf:Property                .
ex:hasSubject              rdf:type    rdf:Property                .
ex:hasBNExclusive          rdf:type    rdf:Property                .
ex:hasDepartment           rdf:type    rdf:Property                .

ex:hasCustomerFavorite     rdfs:domain ex:Book             .
ex:hasCustomerFavorite     rdfs:range  ex:CustomerFavorite .
ex:hasSubject              rdfs:domain ex:Book             .
ex:hasSubject              rdfs:range  ex:Subject          .
ex:hasBNExclusive          rdfs:domain ex:Book                 .
ex:hasBNExclusive          rdfs:range  ex:BNExclusive          .
ex:hasDepartment           rdfs:domain ex:Book             .
ex:hasDepartment           rdfs:range  ex:Department           .
```

Example from FOWST

```
ex:vegetableThaiCurry ex:thaiDishBasedOn ex:CoconutMilk .
ex:sebastian rdf:type ex:AllergicToNuts .
ex:sebastian ex:eats ex:vegetableThaiCurry .

ex:AllergicToNuts rdfs:subClassOf ex:Pitiable .
ex:thaiDishBasedOn rdfs:domain ex:Thai .
ex:thaiDishBasedOn rdfs:range ex:Nutty .
ex:thaiDishBasedOn rdfs:subPropertyOf ex:hasIngredient.
ex:hasIngredient rdf:type rdfs:ContainerMembershipProperty
```

RDFS Limitation

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .
ex:Pizza            rdf:type            ex:Meal .
ex:PizzaMargarita   rdf:type            ex:Pizza .
ex:PizzaMargarita   ex:hasTopping       ex:Tomato .
```

- No Solution for:
  - Pizzas always have at least two toppings. → no cardinality restraints
  - Every pizza from the class *PizzaMargarita* has a *Tomato* topping.
  - Everything having a topping is a pizza.
  - No pizza from the class *PizzaMargarita* has a topping from the class *Meat*

  → no First Order Logic constraints

OWL (Web Ontology Language)

- RDFS is limited in its expression. Not possible to represent complex knowledge.
- In order to model complex knowledge
  - Require expressive representation languages based on formal logic
  - Allows us to do logical reasoning on the knowledge
  - OWL provides this type of knowledge

OWL

- Since 2004 OWL is W3C recommended standard for modeling of ontologies
- Has increased in popularity in many application domains
- In design of OWL, there was a need to find balance between expressivity of the language and scalable reasoning
- Complex language constructs for representing implicit knowledge has a high computational complexity or even undecidability (impossible to construct a single algorithm which produces a “yes” or “no” answer)

OWL

- Statements that can't be modeled in RDF(S) but can be modeled in OWL
  - Every project has at least one participant.
  - Projects are always internal or external projects.
  - Gisela Schillinger and Anne Eberhardt are the secretaries of Rudi Studer.
  - The superior of my superior is also my superior.

OWL

- Users are given a choice between different degrees of expressivity
- Three sublanguages
  - OWL Lite
  - OWL DL (Description Logic)
  - OWL Full

OWL Sublanguages

- OWL Full
  - Contains OWLDL and OWL Lite
  - Only OWL sublanguage containing all of RDFS
  - Very expressive
    - can change meaning of predefined RDF or OWL primitives by applying the language primitives to each other
    - cardinality constraint on the class of all classes, limiting number of classes in the ontology
  - Semantically difficult to understand and work with
  - Undecidable
  - Not supported by software tools

OWL Sublanguages

- OWL DL
  - Contains OWL Lite
  - Decidable
  - Fully supported by most software tools
- OWL Lite
  - Contained in OWL DL and OWL Full
  - Decidable
  - Less expressive

Tradeoff between Expressive Power and Efficient Reasoning Support

- The richer the language is, the more inefficient the reasoning support becomes
- Sometimes it crosses the border of *noncomputability*
- We need a compromise:
  - A language supported by reasonably efficient reasoners
  - A language that can express large classes of ontologies and knowledge.

Reasoning About Knowledge in Ontology Languages

- Class membership
  - If x is an instance of a class C, and C is a subclass of D, then we can infer that x is an instance of D
- Equivalence of classes
  - If class A is equivalent to class B, and class B is equivalent to class C, then A is equivalent to C, too

Reasoning About Knowledge in Ontology Languages

- Consistency
  - X instance of classes A and B, but A and B are disjoint
  - This is an indication of an error in the ontology
- Classification
  - Certain property-value pairs are a sufficient condition for membership in a class A;
  - if an individual x satisfies such conditions, we can conclude that x must be an instance of A

Uses for Reasoning

- Reasoning support is important for
  - checking the consistency of the ontology and the knowledge
  - checking for unintended relationships between classes
  - automatically classifying instances in classes
- Checks like the preceding ones are valuable for
  - designing large ontologies, where multiple authors are involved
  - integrating and sharing ontologies from various sources

Reasoning Support for OWL

- Semantics is a prerequisite for reasoning support
- Formal semantics and reasoning support are usually provided by
  - mapping an ontology language to a known logical formalism
  - using automated reasoners that already exist for those formalisms
- OWL is (partially) mapped on a description logic, and makes use of reasoners such as FaCT, Pellet and RACER
- Description logics are a subset of predicate logic for which efficient reasoning support is possible

Limitations of the Expressive Power of RDF Schema

- Local scope of properties
  - **rdfs:range** defines the range of a property (e.g. eats) for all classes
  - In RDF Schema we cannot declare range restrictions that apply to some classes only
  - E.g. we cannot say that cows eat only plants, while other animals may eat meat, too

Limitations of the Expressive Power of RDF Schema (2)

- Disjointness of classes
  - Sometimes we wish to say that classes are disjoint (e.g. **male** and **female**)
- Boolean combinations of classes
  - Sometimes we wish to build new classes by combining other classes using union, intersection, and complement
  - E.g. **person** is the disjoint union of the classes **male** and **female**

Limitations of the Expressive Power of RDF Schema (3)

- Cardinality restrictions
  - E.g. a person has exactly two parents, a course is taught by at least one lecturer
- Special characteristics of properties
  - Transitive property (like “greater than”)
  - Unique property (like “is mother of”)
  - A property is the inverse of another property (like “eats” and “is eaten by”)

Header of OWL Ontology

- OWL document contains information about namespaces, versioning and annotations
- No direct impact on knowledge but mainly needed for usability
- Every OWL document is an RDF document
- Contains a root element

```
<rdf:RDF
    xmlns    =”http://www.example.org/”
    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:owl=”http://www.w3.org/2002/07/owl#” >
```

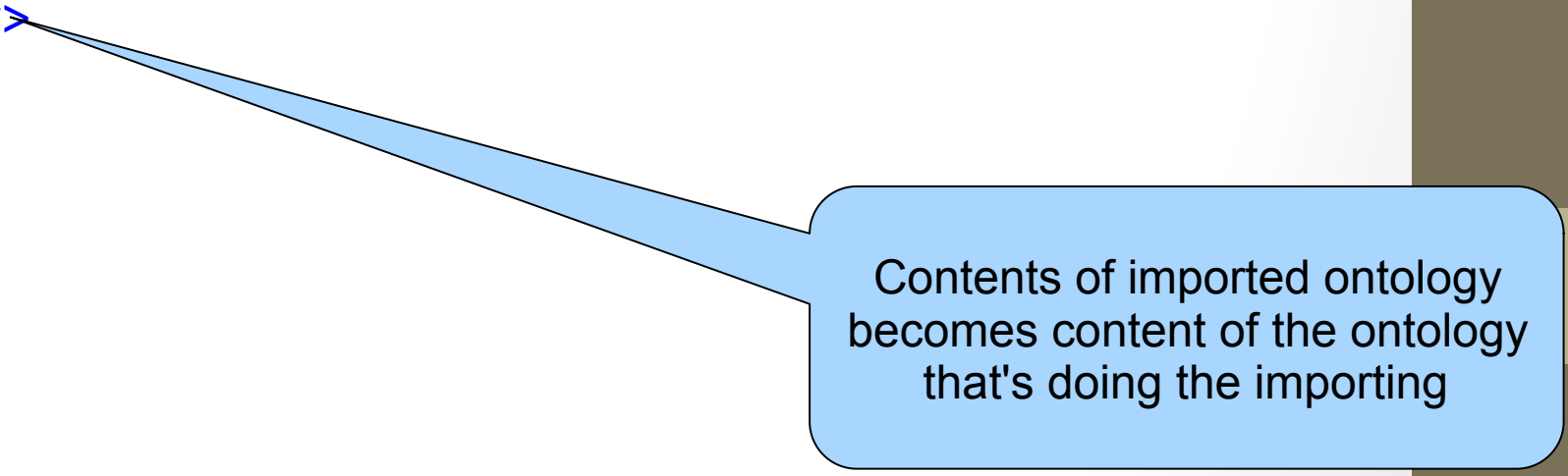
Objects without prefix

Namespace for owl

Header

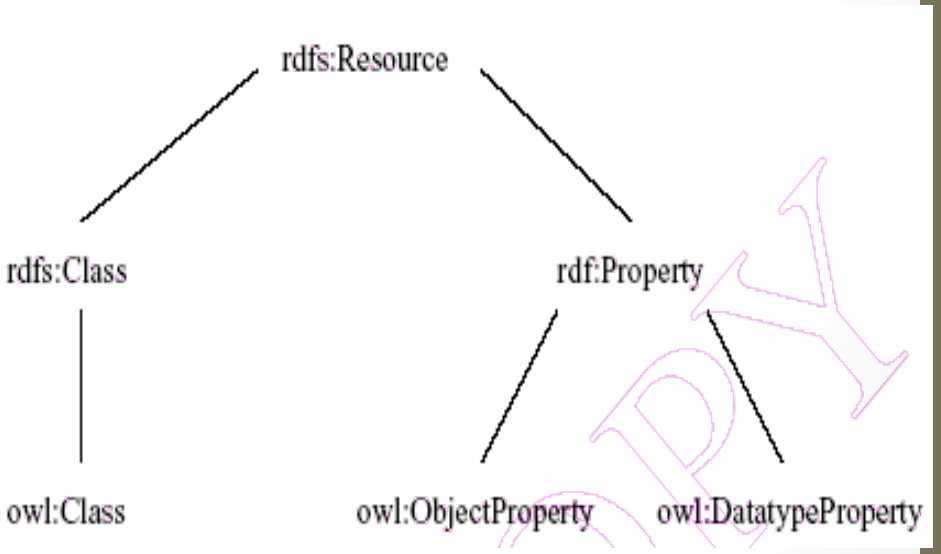
- OWL document may also contain general information about ontology
- owl:Ontology element

```
<owl:Ontology rdf:about=" ">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        SWRC ontology, version of June 2007</rdfs:comment>
    <owl:versionInfo>v0.7.1</owl:versionInfo>
    owl:imports rdf:resource="http://www.example.org/foo"/>
    owl:priorVersion rdf:resource="http://ontoware.org/projects/swrc"/>
</owl:Ontology>
```



OWL Compatibility with RDF Schema

- All varieties of OWL use RDF for their syntax
- Instances are declared as in RDF, using RDF descriptions
- and typing information OWL constructors are specializations of their RDF counterparts



OWL Compatibility with RDF Schema (2)

- Semantic Web design aims at downward compatibility with corresponding reuse of software across the various layers
- The advantage of full downward compatibility for OWL is only achieved for OWL Full, at the cost of computational intractability

Classes, Properties, Individuals

- Basic building blocks of OWL: classes, properties, and individuals
- OWL properties can also be called *roles*
- Classes defined using owl:Class

```
<rdf:Description rdf:about=”Professor”>
    <rdf:type rdf:resource=”&owl;Class”/>
</rdf:Description>
```

  - Equivalent to: <owl:Class rdf:about=”Professor”/>
  - rdf:about assigns the name Professor to the OWL class
  - Also possible to use rdf:ID when this class is being first created
  - But use rdf:about for all subsequent references to that name

Classes

- Two predefined classes
  - owl:Thing
    - is the most general class and has every individual as an instance
  - owl:Nothing
    - Has no instances
- owl:Class is a subclass of rdfs:Class
- As in RDF, individuals can be declared to be instances of classes (class assignment)

```
<rdf:Description rdf:about=”rudiStuder”>
    <rdf:type rdf:resource=”Professor”/>
</rdf:Description>
```

- Equivalent to: <Professor rdf:about=rudiStuder”/>

Properties (Roles)

- Two different kind of properties in OWL: abstract and concrete
- Abstract properties
  - Connect individuals with individuals
  - Also known as object properties
- Concrete properties
  - Connect individuals with data values (elements of datatypes)
  - Also known as datatype properties

Properties

- Both object properties and datatype properties are subclasses of the class rdf:Property

<owl:ObjectProperty rdf:about=”hasAffiliation”/>

<owl:DatatypeProperty rdf:about=”firstName”/>

- hasAffiliation is an object property and expresses which organization a given person is affiliated with
- Second property is concrete and assigns first names to persons
- Properties have domain/range values as in RDFS
- Object properties will have domain/ranges assigned to classes
- Datatype properties will have ranges assigned to literal types

Properties

```
<owl:ObjectProperty rdf:about=”hasAffiliation”>
    <rdfs:domain rdf:resource=”Person”/>
    <rdfs:range rdf:resource=”Organization/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about=”firstName”>
    <rdfs:domain rdf:resource=”Person”/>
    rdfs:range rdf:resource=”&xsd;string”/>
</owl:DatatypeProperty>
```

Properties

- As in RDF, individuals can have assigned roles
- Roles may or may not be functional because there is a way to give two affiliations for an individual

```
<Person rdf:about=”rudiStruder”>
    <hasAffiliation rdf:resource=”aifb”/>
    <hasAffiliation rdf:resource=”ontoprise”/>
    <firstName rdf:datatype=”&xsd;string”>Rudi</firstName>
</Person>
```

Simple Class Relations

- As in RDF(S), subclass relationships can be expressed in OWL with the same

rdfs:subClassOf property

```
<owl:Class rdf:about=”Professor”>
    <rdfs:subClassOf      rdf:resource=”FacultyMember” />
</owl:Class>
```

Simple Class Relations

- Classes are also transitive in OWL which lets us draw simple inferences

```
<owl:Class rdf:about=”Professor”>
    <rdfs:subClassOf rdf:resource=”FacultyMember” />
</owl:Class>
<owl:Class rdf:about=”FacultyMember”>
    <rdfs:subClassOf rdf:resource=”person” />
</owl:Class>
```

- Allow us to infer that Professor is a subclass of Person

Inference with owl:disjointWith

- Disjointness
  - Two classes can be declared to be disjoint using owl:disjointWith property
  - This means that they do not share any individual (intersection between classes is empty)

Inference with owl:disjointWith

- Following inference can be done with disjointness property

```
<owl:Class rdf:about=”Professor”>
    <rdfs:subClassOf rdf:resource=”FacultyMember” />
</owl:Class>
<owl:Class rdf:about=”Book”>
    <rdfs:subClassOf rdf:resource=”Publication” />
</owl:Class>
<owl:Class rdf:about=”FacultyMember”>
    <owl:disjointWith rdf:resource=”Publication” />
</owl:Class>
```

- Allow us to infer that Professor and Book are also disjoint

Inference with owl:EquivalentClass

- Equivalent classes
  - Two classes can be declared to be equivalent by using the owl:equivalentClass property
  - Equivalence can also be achieved by stating that two classes are subclasses of each other

Inference with owl:EquivalentClass

```
<owl:Class rdf:about=”Man”>
    <rdfs:subClassOf rdf:resource=”Person”/>
</owl:Class>
<owl:Class rdf:about=”Person”>
    <owl:equivalentClass rdf:resource=”Human”/>
</owl:Class>
```

- Allows us to infer that man is a subclass of Human

Relations Between Individuals

```
<Book rdf:about=”http://semantic-web-book.org/uri”>
    <author rdf:resource=”markusKroetzch”/>
    <author rdf:resource=”sebastianRudolph”/>
</Book>

<owl:Class rdf:about=”Book”>
    <rdfs:subClassOf rdf:resource=”Publication”/>
</owl:Class>
```

- We can infer that http://semantic-web-book.org/uri is a Publication

Relations Between Individuals

- OWL also allows us to declare that two individuals are the same
- owl:sameAs property allows us to infer this

Relations Between Individuals

```
<Professor rdf:about=”rudiStuder”/>
<rdf:Description rdf:about=”rudiStruder”>
    <owl:sameAs rdf:resource=”professorStuder”/>
</rdf:Description>
```

- We can infer that professorStuder is in the class Professor

Relations Between Individuals

- Unique Name Assumption

```
<rdf:Description rdf:about=”rudiStuder”>
    <owl:sameAs rdf:resource=”professorStuder”/>
</rdf:Description>
```

- Most knowledge representation languages impose Unique Name Assumption: assumed that differently name individuals are different
- OWL allows this to be bypassed by the owl:sameAs property
  - Differently named individuals can denote the same thing
  - Using owl:sameAs allows us to declare this explicitly
  - Also possible to identify this implicitly via inference

Relations Between Individuals

- The owl:sameAs statements are often used in defining mappings between ontologies.
- Unrealistic to assume everybody will use the same name to refer to individuals.
- This would require some grand design, which is contrary to the spirit of the web.

```
<rdf:Description rdf:about="#William_Jefferson_Clinton">
    <owl:sameAs rdf:resource="#BillClinton"/>
</rdf:Description>
```

- Someone may have declared William_Jefferson_Clinton as an individual and this can be equated with his common name BillClinton

Relations Between Individuals

- Possible to declare that individuals are different with owl:differentFrom property
- Can declare that several individuals are mutually different using the rdf:parseType="Collection" property for closed lists

```
<owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
        <Person rdf:about="rudiStuder" />
        <Person rdf:about="dennyVrandecic" />
        <Person rdf:about="peterHaase" />
    </owl:distinctMembers>
</owl:AllDifferent>
```

Closed Classes

<SecretariesOfStuder rdf:about=”giselaSchillinger”/>

<SecretariesOfStuder rdf:about=”anneEberhardt”/>

- giselaSchillinger and annEberhardt are secretaries of Studer
- Does not say anything about whether he has more secretaries or only those two secretaries
- OWL provides closed classes

Closed Classes

```
<owl:Class rdf:about="SecretariesOfStuder">
    <owl:oneOf rdf:parseType="Collection">
        <Person rdf:about="giselaSchillinger"/>
        <Person rdf:about="anneEberhardt" />
    </owl:oneOf>
</owl:Class>
```

- States that giselaSchillinger and anneEberhardt are the only individuals in the class SecretariesOfStuder

Closed Classes

```
<Person rdf:about=”anubriyaAnkolekar” />
<owl:AllDifferent>
    <owl:DistinctMembers rdf:parseType=”Collection”>
        <Person rdf:about=”anneEberhardt”/>
        <Person rdf:about=”giselSchillinger”/>
        <Person rdf:about=”anupriyaAnkolekar”/>
    </owl:distinctMembers>
</owl:AllDifferent>
```

- If we add the following OWL statements, it can also be inferred that anupriyaAnkolekar is not in the class SecretariesOfStuder
- Without the knowledge that individuals are different, it is not possible to infer this
- This knowledge excludes identification of anupriyaAnkolekar with giselaSchillinger or anneEberhardt

Boolean Class Constructors

- So far we have hardly surpassed RDF(S) expressivity
- In order to express more complex knowledge, OWL provides logical class constructors
  - Language elements for conjunction, disjunction and negation
  - Expressed by owl:intersectionOf, owl:unionOf and owl:complementOf
- Constructors allow us to combine atomic classes (class names) to complex classes

owl:intersectionOf

- Conjunction of two classes consists of exactly those objects which belong to both classes

```
<owl:Class rdf:about="SecretariesOfStuder">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Secretaries"/>
        <owl:Class rdf:about="MembersOfStudersGroup"/>
    </owl:intersectionOf>
</owl:Class
```

- Example states that secretariesOfStuder consists of exactly those objects which are both Secretaries and MembersOfStudersGroup
- Example of an inference that can be drawn from this is that all instances of the class SecretariesOfStuder are also in the class Secretaries and MembersOfStudersGroup

owl:unionOf

- Possible to use boolean class constructors with rdfs:subClassOf

```
<owl:Class rdf:about="Professor">
    <rdfs:subClassOf>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="ActivelyTeaching"/>
                <owl:Class rdf:about="Retired"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:subClassOf>
</owl:Class>
```

- Professors are actively teaching or retired
- Retired professor is still actively teaching
- Allows possibility that there are teachers who are not professors
- Every Professor is in at least one of the classes ActivelyTeaching and Retired

owl:complementOf

- Logical negation
- Complement of a class consists of exactly those objects which are not members of the class itself
- Equivalent to the statement made using owl:disjointWith

```
<owl:Class rdf:about="FacultyMember">
    <rdfs:subClassOf>
        <owl:Class>
            <owl:complementOf rdf:resource="Publication"/>
        </owl:Class>
    </rdfs:subClassOf>
</owl:Class>
```

owl:complementOf

- Correct use can be tricky

```
<owl:Class rdf:about=”Male”>
      <owl:complementOf rdfs:resource=”Female”/>
</owl:Class>
<owl:Class rdf:about=”Furniture”>
      <rdfs:subClassOf>
            <owl:Class>
                  <owl:complementOf rdf:resource=”Female”/>
            </owl:Class>
      </rdfs:subClassOf>
</owl:Class>
<furniture rdf:about=”myDesk”>
```

Conclude: myDesk is Male because it is known not to be Female.

owl:complementOf

- Conclude that myDesk is Male because it is known not to be Female.
- Wrong to model it this way because there are things that are neither.
- Better to declare Male and Female to be disjoint
- Or to declare male to be equivalent to the intersection of Human and complement of Female

Role Restrictions

- Another type of logic-based constructors for complex classes
- Role restrictions are constructors involving roles
- Universal Quantifier
  - First role restriction is derived from universal quantifier in predicate logic
  - Defines a class as the set of all object for which the given role only attains values from given class
  - owl:allValuesFrom role restriction

Role Restrictions

```
<owl:Class rdf:about=”Exam”>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource=”hasExaminer”/>
            <owl:allValuesFrom rdf:resource=”Professor”/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- Examiners must always be professors
- All examiners of an exam must be professors

Role Restrictions

- Existential Quantifier
  - Declare that any exam must have at least one examiner
  - Role restriction owl:someValuesFrom is used

```
<owl:Class rdf:about=”Exam”>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource=”hasExaminer”/>
            <owl:someValuesFrom rdf:resource=”Person”/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Role Restrictions

- owl:allValuesFrom
  - We can say something about all of the examiners
- owl:someValuesFrom
  - We can say something about at least one of the examiners