Intelligent Internet Technologies

Lectures 22-23.

OWL Syntax and Specification

Alexandra V. Vitko

KNURE, AI Department, alexandra_vitko@yahoo.com

Three Syntaxes for OWL

- Abstract syntax
- XML syntax
- RDF/XML syntax

Abstract Syntax for OWL

- Corresponds to Description Logics and Frames
- Easier to read and write manually(?)

```
Namespace(a=
   <http://cohse.semanticweb.org/ontologies/people#>)
Ontology(
Class(a:bus_driver complete
   annotation(rdfs:comment "Someone who drives a bus.")
   intersectionOf(restriction(a:drives someValuesFrom
   (a:bus))  a:person))
ObjectProperty(a:drives)
...
)
```

Version of Extended BNF

XML Syntax for OWL

```
<owlx:Ontology
   owlx:name="http://www.example.org/wine"
   xmlns:owlx="http://www.w3.org/2003/05/owl-
   xml">

   <owlx:Annotation>
                ...
   </owlx:Annotation>
</owlx:Ontology>
```

RDF/XML Syntax for OWL

- OWL is part of the Semantic Web
- OWL is an extension of RDF
- RDF applications can parse OWL

```
<rdf:RDF     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#">
  <owl:Ontology rdf:about="http://www.example.org/wine">
              ...
  </owl:Ontology>
</rdf:RDF>
```

OWL doesn’t have

- default values
- arithmetic operations
- string operations
- partial imports
- *some other things*

Namespaces

recommended

<rdf:RDF
xmlns:base="*<your_ontology_URI>*#"
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/2000/10/XMLSchema#">

for simple datatypes

Ontology Header (Example)

```
<?xml version="1.0" ?>
<rdf:RDF xmlns: ...     >
<owl:Ontology rdf:about="http://www.example.org/wine">
   <rdfs:comment>An example OWL ontology</rdfs:comment>
   <owl:priorVersion
        rdf:resource="http://www.example.org/wine-2102.owl"/>
   <owl:imports
        rdf:resource="http://www.example.org/food.owl"/>
   <rdfs:label>Wine Ontology</rdfs:label>
</owl:Ontology>
...
</rdf:RDF>
```

Ontology Header

- **owl:Ontology** collects all meta-data
- **rdf:about** provides a name or reference for the ontology.
  - If **rfd:about=""**, the standard case, the name of the ontology is the base URI of the document
- **owl:priorVersion** gives a reference to the prior version of ontology
- **owl:imports** includes referenced ontology to the current ontology
  - import might fail (!)

Classes

What is a Class?

(*Person, Flower, etc.*)

- Some concept in our mind
- A collection of individuals
- A way to describe a part of the world
- An object in the world (OWL Full)

Simple Classes

<**owl:Class** rdf:ID="Winery"/>

<**owl:Class** rdf:ID="Region"/>

Different namespaces !

Use **rdfs:subClassOf** as usual:

```
<owl:Class rdf:ID="Wine">
    <rdfs:subClassOf
        rdf:resource=“#PotableLiquid"/>
    <rdfs:label xml:lang="en">wine</rdfs:label>
    <rdfs:label xml:lang="fr">vin</rdfs:label>
    ...
</owl:Class>
```

owl:class is not rdfs:class

- Rdfs:class is “class of all classes”
- In DL class can not be treated as individuals (undecidable)
- 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, ...

When is a Class not a Class?

Answer: in OWL Lite & OWL DL, when it's an **Individual** - DL restrictions do not permit **Classes** to be treated as **Individuals**

- So, no "Class, an Individual class, being the Class of all Classes" (as in RDF)
- So, **rdfs:Class** cannot be used in OWL Lite or OWL DL
- **owl:Class** is defined as **rdfs:subClassOf rdfs:Class**
- (But, in OWL Full, they coincide!)

owl:Class

- Subclass of **rdf:Class**
- Better to forget about classes of classes
- Top-most class: **owl:Thing**

Class vs. Individual

- **Class** - simply a name and collection of properties that describe a set of individuals

- **Individual** – a member of classes

- Subclass vs. Instance
  - The **president** most likely is a **Class**
  - The **president of Ukraine** is a natural candidate for an **Individual** (no other similar individuals, unique)
  - However, the president of Ukraine can be also considered as class (representing the role, characteristics) of presidents of Ukraine

Individuals

What are the Individuals?

(*Alexandra Vitko, Bill Klinton, etc.*)

- Objects in the world
- Belong to classes
- Are related to other objects and to data values via properties

Individuals in OWL

- OWL is not only a language for defining ontologies - it is used to define their instances (Individuals)
- An individual is minimally introduced by declaring it to be a member of a class.

Example:

```
<Lecturer rdf:ID=”Vitko”/>
```

Class Lecturer should be defined

Individuals in OWL (2)

- Define individual completely by giving values to the properties of the class it belongs to

Example:

```
<Lecturer rdf:ID="Vitko">
  <name>Alexandra</name>
  <surname>Vitko</surname>
  <activity rdf:resource="#DataMiningTeaching" />
  <activity rdf:resource="#WebTeaching" />
</Lecturer>
```

Class Lecturer should be defined and should have properties name, surname, activity

Defining an owl:Class (1)

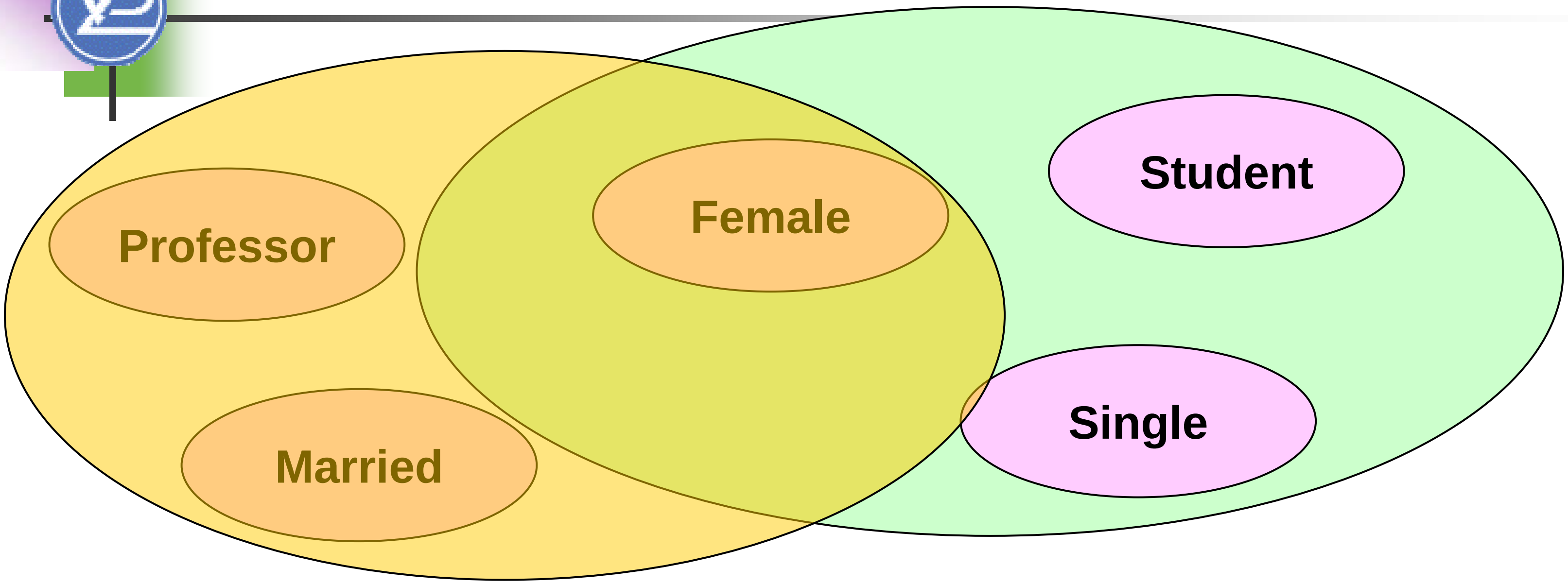
Simple Named Classes

Lite/DL/Full

- By class identifier (simplest):

```
<owl:Class rdf:ID="Lecturer">
  <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>
```

Complex Classes



- Male, Female
- Single, Married, Divorced, Widowed
- Student, Professor
- Single Female Student
- Married Female Professor

Defining Complex Classes

- By enumeration of individuals
- By set operations
- By property restrictions

Defining an owl:Class (2)

Complex Classes

DL/Full

- By enumeration of individuals:

```
<owl:Class rdf:ID=“AIDepartmentLecturer">
  <owl:oneOf rdf:parseType="Collection">
      <Lecturer rdf:about=“#Ryabova" />
      <Lecturer rdf:about=“#Vitko" />
      <Lecturer rdf:about=“#Gvozdinsky"/>
                      ...
  </owl:oneOf>
</owl:Class>
```

Defining an owl:Class (3)

Complex Classes

DL/Full

- By set operations (intersectionOf/unionOf/complementOf):

```
<owl:Class rdf:ID="DepartmentStaff">
   <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#Lecturer" />
         <owl:Class rdf:about="#Researcher" />
         <owl:Class rdf:about="#Engineer" />
   </owl:unionOf>
</owl:Class>
```

Defining an owl:Class (4)

Complex Classes

Lite*/DL/Full

- By property restriction:

```
<owl:Class rdf:ID="Researcher">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#activity" />
         <owl:someValuesFrom
               rdf:resource="#ResearchArea" />
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Restrictions

- Define classes in terms of a restriction that they satisfy with respect to a given property
- **Anonymous: typically included in a class definition 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: Examples

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasChild"/>
  <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">
   3
  </owl:minCardinality>
</owl:Restriction>

<owl:Restriction>
   <owl:onProperty rdf:resource='#hasChild'/>
   <owl:hasValue rdf:datatype="xsd:nonNegativeInteger">
    0
   </owl:hasValue>
</owl:Restriction>
```

Restrictions: More Examples

```
<owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="#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

Axioms in OWL (Equality and Inequality)

- Assertions that are given to be true
- Can be especially powerful in combination with other axioms, which may come from different documents
- Primitives
  - **owl:equivalentClass**
  - **owl:equivalentProperty**
  - **owl:sameAs**
  - **owl:differentFrom**
  - **owl:AllDifferent**

equivalentClass, equivalentProperty

- The property **owl:equivalentClass** is used to indicate that two classes have precisely the same instances!

```
<owl:Class rdf:ID="Student">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#studiesIn" />
      <owl:someValuesFrom rdf:resource="#HEE" />
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Higher Educational Establishment

- To tie together properties in a similar fashion, we use **owl:equivalentProperty**.

Identity between Individuals

Use owl:sameAs

```
<Country rdf:ID="Iran"/>
<Country rdf:ID="Persia">
  <owl:sameAs rdf:resource="#Iran"/>
</Country>
```

In OWL Full owl:sameAs may be used to equate anything: a class and an individual, a property and a class, etc.

Different Individuals

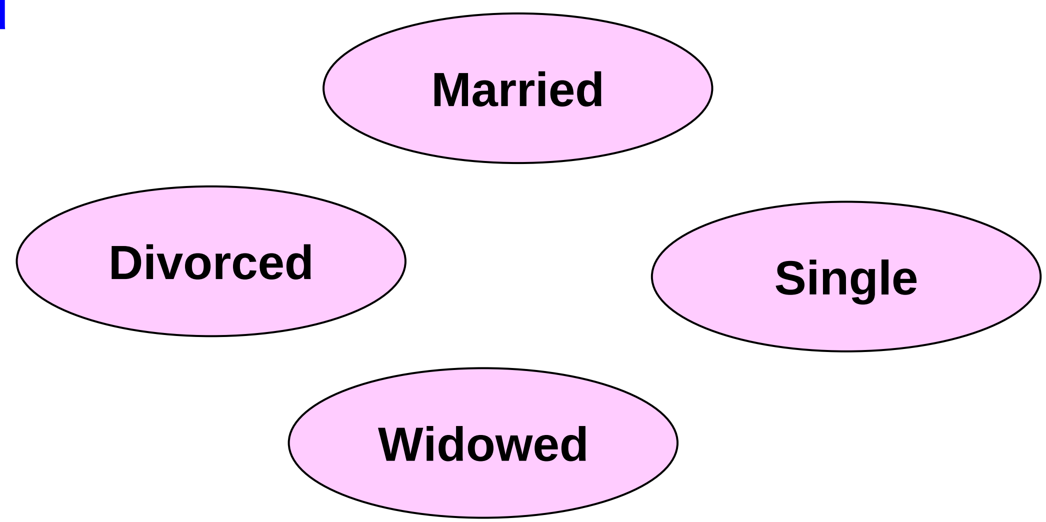
```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Country rdf:ID="Russia"/>
    <Country rdf:ID="Ukraine"/>
    <Country rdf:ID="USA"/>
  </owl:distinctMembers>
</owl:AllDifferent>

<Color rdf:ID="Black"/>
<Color rdf:ID="White">
  <owl:differentFrom rdf:resource="#Black"/>
</Color>
```

Restrictions versus Axioms

- Axioms are global assertions that can be used as the basis for further inference
- Restrictions are constructors
  - When we state that hasChild has a minCardinality of 3, we are
    - Defining the class of persons who have 3 or more children: this class may or may not have any instances
    - Not stating that all persons have 3 or more children
- Often, to achieve the desired effect, we would have to combine restrictions with axioms

Disjoint Classes

- Married disjoint with:
  - Divorced
  - Widowed
  - Single



Disjoint Classes Example

```
<owl:Class rdf:ID="Vegetable">
  <rdfs:subClassOf rdf:resource="#EdibleThing"/>
  <owl:disjointWith rdf:resource="#Meat"/>
  <owl:disjointWith rdf:resource="#Seafood"/>
  <owl:disjointWith rdf:resource="#Fruit"/>
</owl:Class>
```

A common requirement is to define a class as the union of a set of mutually disjoint subclasses.

Properties

What is a Property (*hasChild, age, etc.*)?

- A collection of relationships between individuals and data
- A way of describing a kind of relationship between individuals
- An object in the world (OWL Full)

OWL Properties

- Two types
  - **ObjectProperty** – defines a relation between instances of classes
  - **DatatypeProperty** - relates an instance to an rdfs:Literal or XML Schema datatype

- Both rdfs:subClassOf rdf:Property

Object Property Example

```
<owl:ObjectProperty rdf:ID=”activity">
    <rdfs:domain rdf:resource=“#Person" />
    <rdfs:range rdf:resource=“#ActivityArea" />
</owl: ObjectProperty>
```

Class

Class

Datatype Property Example

```
<owl:DatatypeProperty rdf:ID=”name">
   <rdfs:domain rdf:resource=“#Person" />
   <rdfs:range rdf:resource=“&xsd;string" />
</owl: DatatypeProperty>
```

Class

Simple value

OWL DataTypes

- Full use of XML schema data type definitions
- Create new datatypes as complex classes

Examples

- Define a type age that must be a non-negative integer
- Define a type clothing size that is an enumeration “small”, “medium”, “large”

Recommended datatypes to be used with OWL

xsd:string	xsd:normalizedString	xsd:boolean	
xsd:decimal	xsd:float	xsd:double	
xsd:integer	xsd:nonNegativeInteger	xsd:positiveInteger	
xsd:nonPositiveInteger	xsd:negativeInteger		
xsd:long	xsd:int	xsd:short	xsd:byte
xsd:unsignedLong	xsd:unsignedInt	xsd:unsignedShort	xsd:unsignedByte
xsd:hexBinary	xsd:base64Binary		
xsd:dateTime	xsd:time	xsd:date	xsd:gYearMonth
xsd:gYear	xsd:gMonthDay	xsd:gDay	xsd:gMonth
xsd:anyURI	xsd:token	xsd:language	
xsd:NMTOKEN	xsd:Name	xsd:NCName	

Like rdf:Property

- Can be arranged in a hierarchy
- Multiple domains mean that the domain of the property is the intersection of the identified classes (and similarly for range).

Unlike rdf:Property

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdfs:domain
   rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  <rdfs:range rdf:resource="#Region"/>
</owl:ObjectProperty>
```

Top-most Class

Property Characteristics

- Transitive Property
- Symmetric Property
- Functional Property
- Inverse Property
- Inverse Functional Property

Transitive Property

$X \rightarrow p_1 \rightarrow Y$

$Y \rightarrow p_1 \rightarrow Z$

implies $X \rightarrow p_1 \rightarrow Z$

Examples:

- *located_in*
- *part_of*

Transitive Property Example

```
<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="Alekseevka">
   <locatedIn rdf:resource="#Kharkov" />
</Region>

<Region rdf:ID="Kharkov">
   <locatedIn rdf:resource="#Ukraine" />
</Region>
```

Symmetric Property

X → p_1 → Y

implies X ← p_1 ← Y

Examples:

- *friendOf*
- *neighbourOf*

Symmetric Property Example

```
<owl:ObjectProperty rdf:ID="friendOf">
  <rdf:type
   rdf:resource="&owl;SymmetricProperty"/>
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="#Person" />
</owl:ObjectProperty>
```

Functional Properties

X → p1 → Y

X → p1 → Z

imply Z is the same as Y

(they describe the same)

Example:

- *hasMother*

Functional Property Example

```
<owl:ObjectProperty rdf:ID=“hasMother">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource=“#Person" />
  <rdfs:range rdf:resource#Woman" />
</owl:ObjectProperty>
```

```
<Person rdf:ID=“IvanovV">
    <hasMother rdf:resource=“#IvanovaM" / >
</Person>

<Person rdf:ID=“IvanovV">
    <hasMother rdf:resource=“#IvanovaMaria" >
</Person>
```

IvanovaM
=
IvanovaMaria

Inverse Property

$X \rightarrow p_1 \rightarrow Y$

implies $Y \rightarrow p2 \rightarrow X$

Example:

- *isChildOf and isParentOf*

Symmetric Property Example

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

<owl:ObjectProperty rdf:ID="isParentOf">
    <owl:inverseOf rdf:resource="#isChildOf"/>
</owl:ObjectProperty>
```

Inverse Functional Property

Y → p1 → A

Z → p1 → A

imply Z is the same as Y

(they describe the same)

Example:

- *hasIdentificationNumber*

Inverse Functional Property Example

```
<owl:ObjectProperty rdf:ID=“hasIDNumber">
  <rdf:type
   rdf:resource="&owl;InverseFunctionalProperty"/>
  <rdfs:domain rdf:resource=“#Person" />
  <rdfs:range rdf:resource=“&xsd;integer" />
</owl:ObjectProperty>
```

```
<Person rdf:ID=“VanyaIvanov">
    <hasIDNumber>1234567890</hasIdNumber>
</Person>

<Person rdf:ID=“IvanovV">
    <hasIDNumber>1234567890</hasIdNumber>
</Person>
```

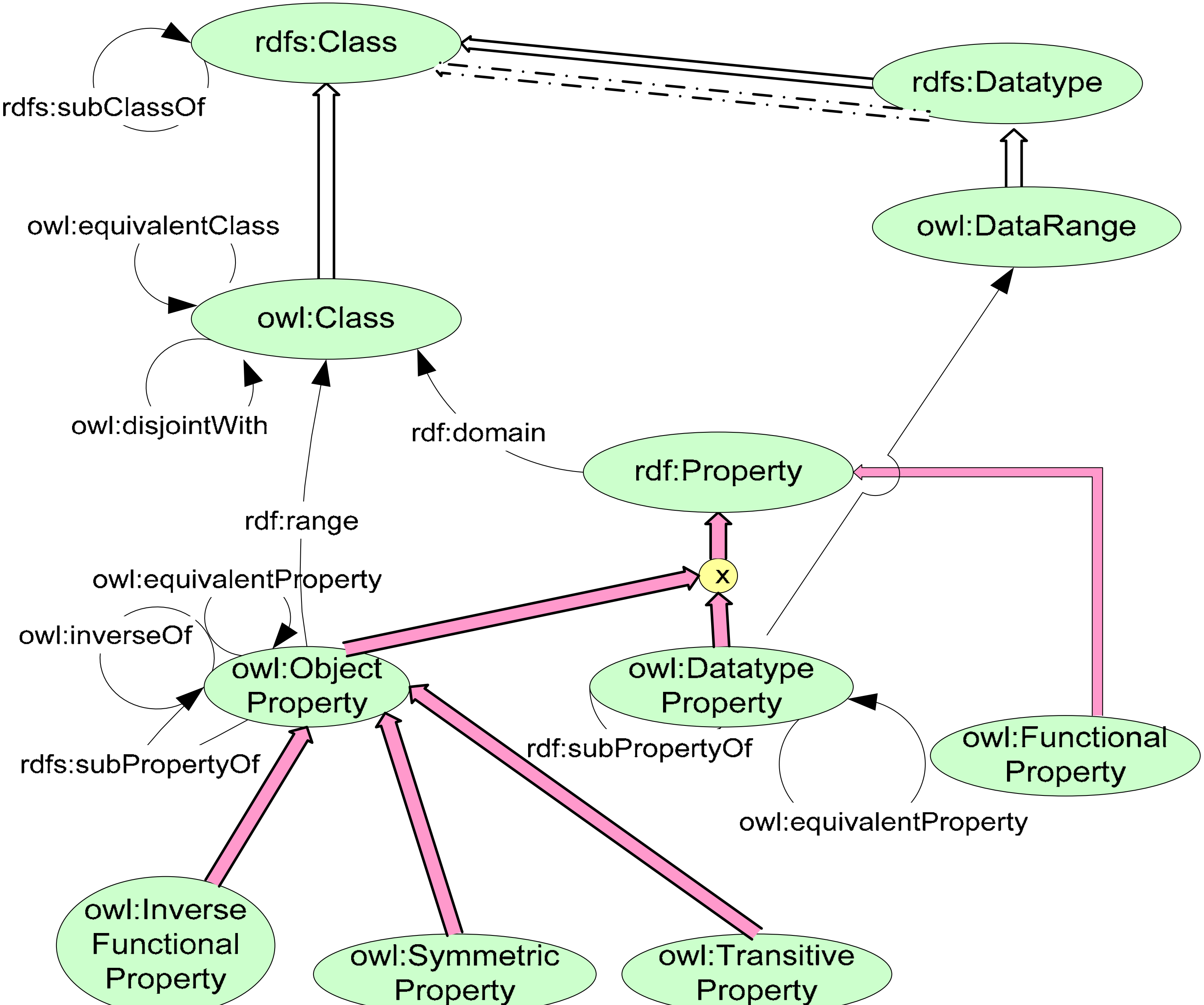
VanyaIvanov
=
IvanovV

Property Restrictions

- owl:allValuesFrom
- owl:someValuesFrom
- hasValue
- cardinality:
  - minCardinality
  - maxCardinality
  - cardinality (when min=max)

Examples on slides 26-27

OWL Entities and Relationships



Conclusions

- OWL is more expressive than RDF(S)
- OWL evolved from DAML+OIL
- OWL is potentially the most important knowledge representation language we've yet seen
- It could be the "last word" in Web knowledge representation similar to how HTML came to dominate the field of hypertext markup