Chapter 4
OWL

Based on slides from Grigoris Antoniou and Frank van Harmelen

Outline

1. A bit of history
2. Basic Ideas of OWL
3. The OWL Language
4. Examples
5. The OWL Namespace
6. Future Extensions

The OWL Family Tree

A Brief History of OWL: SHOE

- Simple HTML Ontology Extensions
  SHOE allows World-Wide Web authors to annotate their pages with ontology-based knowledge about page contents. We present examples showing how the use of SHOE can support a new generation of knowledge-based search and knowledge discovery tools that operate on the World-Wide Web.
- Supported adding “semantic” tags defined in an ontology plus prolog-like rules to web pages.
A Brief History of OWL: SHOE

Developed by group of (largely) European researchers (several from EU OntoKnowledge project)
- Based on frame-based language
- Strong emphasis on formal rigour.
- Semantics in terms of Description Logics
- RDFS based syntax

A Brief History of OWL: OIL

Developed by group of (largely) European researchers (several from EU OntoKnowledge project)
- Based on frame-based language
- Strong emphasis on formal rigour.
- Semantics in terms of Description Logics
- RDFS based syntax

A Brief History of OWL: DAML-ONT

- Developed by DARPA DAML Program.
  - Largely US based researchers
- Extended RDFS with constructors from OO and frame-based languages
- Rather weak semantic specification
  - Problems with machine interpretation
  - Problems with human interpretation

A Brief History of OWL: DAML+OIL

- Merging of DAML-ONT and OIL
- Basically a DL with an RDFS-based syntax.
- Development was carried out by “Joint EU/US Committee on Agent Markup Languages”
- Extends (“DL subset” of) RDF
- Submitted to W3C as basis for standardisation
  - Web-Ontology (WebOnt) Working Group formed
A Brief History of OWL: OWL

- W3C Recommendation (February 2004)
- Based largely on the March 2001 DAML+OIL specification
- Well defined RDF/XML serializations
- Formal semantics
  - First Order
  - Relationship with RDF
- Comprehensive test cases for tools/implementations
- Growing industrial take up.

OWL 1.1

- Is an extension of OWL
  - Addresses deficiencies identified by users and developers (at OWLED workshop)
- Is based on more expressive DL: SROIQ
  - OWL is based on SHOIN
- W3C working group chartered
  - Develop recommendation to be voted on in April 2009
- Supported by popular OWL tools
  - Protégé, Swoop, TopBraid, FaCT++, Pellet

Outline

1. A bit of history
2. Basic Ideas of OWL
3. The OWL Language
4. Examples
5. The OWL Namespace
6. Future Extensions
Requirements for Ontology Languages

- Ontology languages allow users to write explicit, formal conceptualizations of domain models
- The main requirements are:
  - a well-defined syntax
  - efficient reasoning support
  - a formal semantics
  - sufficient expressive power
  - convenience of expression

Expressive Power vs Efficient Reasoning

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

Kinds of Reasoning about Knowledge

- 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
- 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 these 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, RACER and Pellet
- Description logics are a subset of predicate logic for which efficient reasoning support is possible

RDFS’s Expressive Power Limitations

- 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

RDFS’s Expressive Power Limitations

- 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`

RDFS’s Expressive Power Limitations

- 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”)
Combining OWL with RDF Schema

- Ideally, OWL would extend RDF Schema
  - Consistent with the layered architecture of the Semantic Web
- **But** simply extending RDF Schema would work against obtaining expressive power and efficient reasoning
  - Combining RDF Schema with logic leads to uncontrollable computational properties

Three Species of OWL

- W3C’s Web Ontology Working Group defined OWL as three different sublanguages:
  - OWL Full
  - OWL DL
  - OWL Lite
- Each sublanguage geared toward fulfilling different aspects of requirements

OWL Full

- It uses all the OWL languages primitives
- It allows the combination of these primitives in arbitrary ways with RDF and RDF Schema
- OWL Full is fully upward-compatible with RDF, both syntactically and semantically
- OWL Full is so powerful that it’s undecidable
  - No complete (or efficient) reasoning support

Soundness and completeness

- A sound reasoner only makes conclusions that logically follow from the input, i.e., all of it’s conclusions are correct
  - We almost always require our reasoners to be sound
- A complete reasoner can make all of the conclusions that logically follow from the input
  - We can not guarantee complete reasoners for full FOL and many subsets
OWL DL

- OWL DL (Description Logic) is a sublanguage of OWL Full that restricts application of the constructors from OWL and RDF
  - Application of OWL's constructors' to each other is disallowed
  - Therefore it corresponds to a well studied description logic
- OWL DL permits efficient reasoning support
- **But** we lose full compatibility with RDF:
  - Not every RDF document is a legal OWL DL document.
  - Every legal OWL DL document is a legal RDF document.

OWL Lite

- An even further restriction limits OWL DL to a subset of the language constructors
  - E.g., OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality.
- The advantage of this is a language that is easier to
  - grasp, for users
  - implement, for tool builders
- The disadvantage is restricted expressivity

Upward Compatibility for OWL Species

- Every legal OWL Lite ontology is a legal OWL DL ontology
- Every legal OWL DL ontology is a legal OWL Full ontology
- Every valid OWL Lite conclusion is a valid OWL DL conclusion
- Every valid OWL DL conclusion is a valid OWL Full conclusion

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 specialisations of their RDF counterparts
OWL Compatibility with RDF Schema

- 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

OWL Syntactic Varieties

- OWL builds on RDF and uses RDF’s XML-based syntax
- Other syntactic forms for OWL have also been defined:
  - An alternative, more readable XML-based syntax
  - An abstract syntax, that is much more compact and readable than the XML languages
  - A graphic syntax based on the conventions of UML

OWL XML/RDF Syntax: Header

```xml
<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#"
  xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">```

- OWL documents are RDF documents
- and start with a typical declaration of namespaces
- The W3C recommendation for owl has the namespace http://www.w3.org/2002/07/owl#
owl:Ontology

<owl:Ontology rdf:about=""/>
<rdfs:comment>Example OWL ontology</rdfs:comment>
<owl:priorVersion rdf:resource="http://www.-mydomain.org/uni-ns-old"/>
<owl:imports rdf:resource="http://www.-mydomain.org/-persons"/>
<rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>

- owl:imports, a transitive property, indicates that the document commits to all of the terms as defined in its target.
- owl:priorVersion points to an earlier version of this document

Why Separate Classes & Datatypes?

- Philosophical reasons:
  - Datatypes structured by built-in predicates
  - Not appropriate to form new datatypes using ontology language
- Practical reasons:
  - Note: Java does this, distinguishing classes from primitive datatypes
  - Ontology language remains simple and compact
  - Semantic integrity of ontology language not compromised
  - Implementability not compromised — can use hybrid reasoner
    - Only need sound and complete decision procedure for:
      - $d_1 \lor \ldots \lor d_n$, where $d_i$ is a (possibly negated) datatype

OWL Classes

<owl:Class rdf:about="#associateProfessor">
<owl:disjointWith rdf:resource="#professor"/>
<owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>

- Classes are defined using owl:Class
- owl:Class is a subclass of rdfs:Class
- Owl:Class is disjoint with datatypes
- Disjointness is defined using owl:disjointWith
  - Two disjoint classes are can share no instances

OWL Classes

<owl:Class rdf:ID="faculty">
<owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>

- owl:equivalentClass defines equivalence of classes
- owl:Thing is the most general class, which contains everything
  - i.e., every owl class is rdf:subClassOf owl:Thing
- owl:Nothing is the empty class
  - i.e., owl:Nothing is rdf:subClassOf every owl class
In OWL there are two kinds of properties

- **Object properties** relate objects to other objects
  - `owl:DatatypeProperty`
  - E.g. `is-TaughtBy`, `supervises`

- **Data type properties** relate objects to datatype values
  - `owl:ObjectProperty`
  - E.g. `phone`, `title`, `age`, etc.

OWL Object Properties

- Typically user-defined data types

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
  <owl:domain rdf:resource="#course"/>
  <owl:range rdf:resource="#academicStaffMember"/>
  <rdfs:subPropertyOf rdf:resource="#involves"/>
</owl:ObjectProperty>
```

Datatype Properties

- OWL uses XML Schema data types, exploiting the layered architecture of the Semantic Web

```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XLMSchema#nonNegativeInteger"/>
  <rdfs:domain rdf:resource="foaf:Person"/>
</owl:DatatypeProperty>
```

Inverse Properties

```
<owl:ObjectProperty rdf:ID="teaches">
  <rdfs:range rdf:resource="#course"/>
  <rdfs:domain rdf:resource="#academicStaffMember"/>
  <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

A partial list of axioms:

- `(?P owl:inverseOf ?Q. ?P @has rdfs:domain ?C) => (?Q rdfs:range ?C).` (O_B)
Equivalent Properties

```
<owl:equivalentProperty
  <owl:ObjectProperty rdf:ID="lecturesIn">
    <owl:equivalentProperty rdf:resource="#teaches"/>
  </owl:ObjectProperty>

- Two properties have the same property extension
- Axioms
    <=>  {?A owl:equivalentProperty ?B}.
```

Property Restrictions

- In OWL we can declare that the class C satisfies certain conditions
  - All instances of C satisfy the conditions
- This is equivalent to saying that C is subclass of a class C', where C collects all objects that satisfy the conditions
  - C' can remain anonymous
- Example:
  - People whose sex is male and have at least one child whose sex is female and whose age is six
  - Things with exactly two arms and two legs

```
Property Restrictions

- The owl:Restriction element describes such a class
- This element contains an owl:onProperty element and one or more restriction declarations
- One type defines cardinality restrictions (at least one, at most 3, …)
- The other type defines restrictions on the kinds of values the property may take
  - owl:allValuesFrom specifies universal quantification
  - owl:hasValue specifies a specific value
  - owl:someValuesFrom specifies existential quantification

owl:allValuesFrom

- Describe a class where all of the values of a property match some requirement
- E.g., Math courses taught by professors.
```

```
<!-- First year courses that are taught by professors -->
<owl:Class rdf:about="#firstYearCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:allValuesFrom rdf:resource="#Professor"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
**owl:hasValue**

- Describe a class with a particular value for a property.
- E.g., Math courses taught by Professor Longhair.

```xml
<owl:Class rdf:about="#mathCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource= "#isTaughtBy"/>
      <owl:hasValue rdf:resource= "#949352"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

**Cardinality Restrictions**

- We can specify minimum and maximum number using `owl:minCardinality` & `owl:maxCardinality`
  - Courses with fewer than 10 students
  - Courses with between 10 and 100 students
  - Courses with more than 100 students
- It is possible to specify a precise number by using the same minimum and maximum number
  - Courses with exactly seven students
- For convenience, OWL offers also `owl:cardinality`
  - E.g., exactly N

```xml
<owl:Class rdf:about="#course">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">2</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

**owl:someValuesFrom**

- Describe a class based on a requirement that it must have at least one value for a property matching a description.
- E.g., Academic staff members who teach an undergraduate course.

```xml
<owl:Class rdf:about="#academicStaffMember">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#teaches"/>
      <owl:someValuesFrom rdf:resource="#undergraduateCourse"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

**Cardinality Restrictions**

- E.g. courses taught be at least two people.

```xml
<owl:Class rdf:about="#course">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">2</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
Special Properties

- **owl:TransitiveProperty** (transitive property)
  - E.g. “has better grade than”, “is ancestor of”
- **owl:SymmetricProperty** (symmetry)
  - E.g. “has same grade as”, “is sibling of”
- **owl:FunctionalProperty** defines a property that has at most one value for each object
  - E.g. “age”, “height”, “directSupervisor”
- **owl:InverseFunctionalProperty** defines a property for which two different objects cannot have the same value

Boolean Combinations

- We can combine classes using Boolean operations (union, intersection, complement)
- Negation is introduced by the complementOf
- E.g., courses not taught by staff members
  ```xml
  <owl:Class rdf:about="#course">
    <owl:Restriction>
      <owl:onProperty rdf:resource="#teaches"/>
      <owl:allValuesFrom>
        <owl:complementOf rdf:resource="#staffMember"/>
      </owl:allValuesFrom>
    </owl:Restriction>
    <owl:subClassOf/>
  </owl:Class>
  ```
- The new class is not a subclass of the union, but rather equal to the union
  - We have stated an equivalence of classes
- E.g., university people is the union of staff members and students
  ```xml
  <owl:Class rdf:ID="peopleAtUni">  
    <owl:unionOf rdf:parseType="Collection">  
      <owl:Class rdf:about="#staffMember"/>  
      <owl:Class rdf:about="#student"/>  
    </owl:unionOf>
    <owl:subClassOf/>
  </owl:Class>
  ```
**Boolean Combinations**

*E.g., CS faculty is the intersection of faculty and things that belongTo the CS Department.*

```
<owl:Class rdf:ID="facultyInCS">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#faculty"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#belongsTo"/>
      <owl:hasValue rdf:resource="#CSDepartment"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

**Nesting of Boolean Operators**

*E.g., administrative staff are staff members who are not faculty or technical staff members*

```
<owl:Class rdf:ID="adminStaff">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#staffMember"/>
    <owl:complementOf>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#faculty"/>
        <owl:Class rdf:about="#techSupportStaff"/>
      </owl:unionOf>
    </owl:complementOf>
  </owl:intersectionOf>
</owl:Class>
```

**Enumerations with owl:oneOf**

*E.g., a thing that is either Monday, Tuesday, …*

```
<owl:oneOf rdf:parseType="Collection">
  <owl:Thing rdf:about="#Monday"/>
  <owl:Thing rdf:about="#Tuesday"/>
  <owl:Thing rdf:about="#Wednesday"/>
  <owl:Thing rdf:about="#Thursday"/>
  <owl:Thing rdf:about="#Friday"/>
  <owl:Thing rdf:about="#Saturday"/>
  <owl:Thing rdf:about="#Sunday"/>
</owl:oneOf>
```

**Declaring Instances**

*Instances of classes are declared as in RDF, as in these examples*

```
<rdf:Description rdf:ID="949352">
  <rdf:type rdf:resource="#academicStaffMember"/>
</rdf:Description>
<academicStaffMember rdf:ID="949352">
  <uni:age rdf:datatype="&xsd;integer">39</uni:age>
</academicStaffMember>
```
No Unique-Names Assumption

- OWL does not adopt the unique-names assumption of database systems
  - That two instances have a different name or ID does not imply that they are different individuals
- Suppose we state that each course is taught by at most one staff member, and that a given course is taught by #949318 and is taught by #949352
  - An OWL reasoner does not flag an error
  - Instead it infers that the two resources are equal

Distinct Objects

To ensure that different individuals are indeed recognized as such, we must explicitly assert their inequality:

```xml
<lecturer rdf:about="949318">
  <owl:differentFrom rdf:resource="949352"/>
</lecturer>
```

Distinct Objects

- OWL provides a shorthand notation to assert the pairwise inequality of all individuals in a given list

```xml
<owl:allDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <lecturer rdf:about="949318"/>
    <lecturer rdf:about="949352"/>
    <lecturer rdf:about="949111"/>
  </owl:distinctMembers>
</owl:allDifferent>
```

Data Types in OWL

- XML Schema provides a mechanism to construct user-defined data types
  - E.g., the data type of `adultAge` includes all integers greater than 18
- Such derived data types cannot be used in OWL
  - The OWL reference document lists all the XML Schema data types that can be used
  - These include the most frequently used types such as `string`, `integer`, `Boolean`, `time`, and `date`. 
Versioning Information

- `owl:priorVersion` indicates earlier versions of the current ontology
  - No formal meaning, can be exploited for ontology management
- `owl:versionInfo` generally contains a string giving information about the current version, e.g. keywords

Combination of Features

- In different OWL languages there are different sets of restrictions regarding the application of features
- In OWL Full, all the language constructors may be used in any combination as long as the result is legal RDF

Restriction of Features in OWL DL

- **Vocabulary partitioning**
  - Any resource is allowed to be only a class, a data type, a data type property, an object property, an individual, a data value, or part of the built-in vocabulary, and not more than one of these
- **Explicit typing**
  - The partitioning of all resources must be stated explicitly (e.g. a class must be declared if used in conjunction with `rdfs:subClassOf`)
Restriction of Features in OWL DL

- **Property Separation**
  - The set of object properties and data type properties are disjoint
  - Therefore the following can never be specified for data type properties:
    - `owl:inverseOf`
    - `owl:FunctionalProperty`
    - `owl:InverseFunctionalProperty`
    - `owl:SymmetricProperty`

- **No transitive cardinality restrictions**
  - No cardinality restrictions may be placed on transitive properties
  - e.g., people with more than 5 ancestors

- **Restricted anonymous classes**
  Anonymous classes are only allowed to occur as:
  - the domain and range of either `owl:equivalentClass` or `owl:disjointWith`
  - the range (but not the domain) of `rdfs:subClassOf`

Restriction of Features in OWL Lite

- Restrictions of OWL DL and more
  - `owl:oneOf`, `owl:disjointWith`, `owl:unionOf`, `owl:complementOf` and `owl:hasValue` are not allowed
  - Cardinality statements (minimal, maximal, and exact cardinality) can only be made on the values 0 or 1
  - `owl:equivalentClass` statements can no longer be made between anonymous classes but only between class identifiers

Inheritance in Class Hierarchies

- **Range restriction:** Courses must be taught by academic staff members only
- Ben Bitdiddle is a professor
- He inherits the ability to teach from the class of academic staff members
- This is done in RDF Schema by fixing the semantics of “is a subclass of”
  - It is not up to an application (RDF processing software) to interpret “is a subclass of”
Outline

1. A bit of history
2. Basic Ideas of OWL
3. The OWL Language
4. Examples
5. The OWL Namespace
6. Future Extensions

African Wildlife Ontology: Classes

African Wildlife: Schematic Representation

Branches are parts of trees

African Wildlife: Properties

<owl:TransitiveProperty rdf:ID="is-part-of"/>
<owl:ObjectProperty rdf:ID="eats">
    <rdfs:domain rdf:resource="#animal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="eaten-by">
    <owl:inverseOf rdf:resource="#eats"/>
</owl:ObjectProperty>
African Wildlife: Plants and Trees

<owl:Class rdf:ID="plant">
  <rdfs:comment>Plants are disjoint from animals. </rdfs:comment>
  <owl:disjointWith="#animal"/>
</owl:Class>

<owl:Class rdf:ID="tree">
  <rdfs:comment>Trees are a type of plant. </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#plant"/>
</owl:Class>

An African Wildlife: Branches

<owl:Class rdf:ID="branch">
  <rdfs:comment>Branches are parts of trees. </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#is-part-of"/>
      <owl:allValuesFrom rdf:resource="#tree"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

African Wildlife: Leaves

<owl:Class rdf:ID="leaf">
  <rdfs:comment>Leaves are parts of branches. </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#is-part-of"/>
      <owl:allValuesFrom rdf:resource="#branch"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

African Wildlife: Carnivores

<owl:Class rdf:ID="carnivore">
  <rdfs:comment>Carnivores are exactly those animals that eat also animals. </rdfs:comment>
  <owl:intersectionOf rdf:parsetype="Collection">
    <owl:Class rdf:about="#animal"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#eats"/>
      <owl:someValuesFrom rdf:resource="#animal"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
African Wildlife: Herbivores

<owl:Class rdf:ID="herbivore">
  <rdfs:comment>
  Herbivores are exactly those animals that eat only plants or parts of plants.
  </rdfs:comment>
  <rdfs:comment>
  Try it out! See book for code.
  </rdfs:comment>
</owl:Class>

African Wildlife: Giraffes

<owl:Class rdf:ID="giraffe">
  <rdfs:comment>Giraffes are herbivores, and they eat only leaves.</rdfs:comment>
  <rdfs:subClassOf rdf:type="#herbivore"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#eats"/>
    <owl:allValuesFrom rdf:resource="#leaf"/>
  </owl:Restriction>
</owl:Class>

African Wildlife: Lions

<owl:Class rdf:ID="lion">
  <rdfs:comment>Lions are animals that eat only herbivores.</rdfs:comment>
  <rdfs:subClassOf rdf:type="#carnivore"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#eats"/>
      <owl:allValuesFrom rdf:resource="#herbivore"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

African Wildlife: Tasty Plants

owl:Class rdf:ID="tasty-plant">
  <rdfs:comment>Plants eaten both by herbivores and carnivores</rdfs:comment>
  <rdfs:comment>
  Try it out! See book for code.
  </rdfs:comment>
</owl:Class>
Printer Ontology – Class Hierarchy

- <owl:Class rdf:ID="product">
  <rdfs:comment>Products form a class. </rdfs:comment>
  </owl:Class>

- <owl:Class rdf:ID="padid">
  <rdfs:comment>Printing and digital imaging devices form a subclass of products. </rdfs:comment>
  <rdfs:label>Device</rdfs:label>
  <rdfs:subClassOf rdf:resource="#product"/>
  </owl:Class>

Printer Ontology – HP Products

- <owl:Class rdf:ID="hpProduct">
  <owl:intersectionOf>
    <owl:Class rdf:about="#product"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#manufactured-by"/>
      <owl:hasValue>
        <xsd:string rdf:value="Hewlett Packard"/>
      </owl:hasValue>
    </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

Printer Ontology – Printers & Personal Printers

- <owl:Class rdf:ID="printer">
  <rdfs:comment>Printers are printing and digital imaging devices. </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#padid"/>
  </owl:Class>

- <owl:Class rdf:ID="personalPrinter">
  <rdfs:comment>Printers for personal use form a subclass of printers. </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#printer"/>
  </owl:Class>
HP LaserJet 1100se Printers

<owl:Class rdf:ID="1100se">
  <rdfs:comment>1100se printers belong to the 1100 series and cost $450.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#1100series"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price"/>
      <owl:hasValue><xsd:integer rdf:value="450"/></owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

A Printer Ontology – Properties

<owl:DatatypeProperty rdf:ID="manufactured-by">
  <rdfs:domain rdf:resource="#product"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="printingTechnology">
  <rdfs:domain rdf:resource="#printer"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

Outline

1. A bit of history
2. Basic Ideas of OWL
3. The OWL Language
4. Examples
5. The OWL Namespace
6. Future Extensions

OWL in OWL

- We present a part of the definition of OWL in terms of itself
- The following captures some of OWL’s meaning in OWL
  - It does not capture the entire semantics
  - A separate semantic specification is necessary
- The URI of the OWL definition is defined as the default namespace
Classes of Classes (Metaclasses)

- The class of all OWL classes is itself a subclass of the class of all RDF Schema classes:
  
  ```xml
  <rdfs:Class rdf:ID="Class">
    <rdfs:label>Class</rdfs:label>
    <rdfs:subClassOf rdf:resource="#rdfsClass"/>
  </rdfs:Class>
  ```

Metaclasses – Thing and Nothing

- **Thing** is most general object class in OWL
- **Nothing** is most specific class: the empty object class
- The following relationships hold:

  \[
  \text{Thing} = \text{Nothing} \cup \text{Nothing} \\
  \text{Nothing} = \text{Thing} = \text{Nothing} \cup \text{Nothing} = \text{Nothing} \setminus \text{Nothing} = \emptyset
  \]

Class and Property Equivalences

```xml
<Class rdf:ID="Thing">
  <rdfs:label>Thing</rdfs:label>
  <unionOf rdf:parseType="Collection">
    <Class rdf:about="#Nothing"/>
    <Class>
      <complementOf rdf:resource="#Nothing"/>
    </Class>
  </unionOf>
</Class>

<Class rdf:ID="Nothing">
  <rdfs:label>Nothing</rdfs:label>
  <complementOf rdf:resource="#Thing"/>
</Class>
```

```xml
<rdf:Property rdf:ID="EquivalentClass">
  <rdfs:label>EquivalentClass</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#rdfsSubClassOf"/>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="EquivalentProperty">
  <rdfs:label>EquivalentProperty</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#rdfsSubPropertyOf"/>
</rdf:Property>
```
Class Disjointness

<rdf:Property rdf:ID="disjointWith">
  <rdfs:label>disjointWith</rdfs:label>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

Equality and Inequality

- Equality and inequality can be stated between arbitrary things
  - In OWL Full this statement can also be applied to classes
- Properties sameIndividualAs, sameAs and differentFrom

Equality and Inequality

<rdf:Property rdf:ID="sameIndividualAs">
  <rdfs:domain rdf:resource="#Thing"/>
  <rdfs:range rdf:resource="#Thing"/>
</rdf:Property>

<rdf:Property rdf:ID="sameAs">
  <EquivalentProperty rdf:resource="#sameIndividualAs"/>
</rdf:Property>

Union and Intersection of Classes

- Build a class from a list, assumed to be a list of other class expressions

<rdf:Property rdf:ID="unionOf">
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>
Restriction Classes

- Restrictions in OWL define the class of those objects that satisfy some attached conditions

```xml
<rdfs:Class rdf:ID="Restriction">
    <rdfs:label>Restriction</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Class"/>
</rdfs:Class>
```

Restriction Properties

- All the following properties (onProperty, allValuesFrom, minCardinality, etc.) are only allowed to occur within a restriction definition
  - Their domain is owl:Restriction, but they differ with respect to their range

```xml
<rdf:Property rdf:ID="onProperty">
    <rdfs:label>onProperty</rdfs:label>
    <rdfs:domain rdf:resource="#Restriction"/>
    <rdfs:range rdf:resource="&rdfs;Property"/>
</rdf:Property>

<rdf:Property rdf:ID="allValuesFrom">
    <rdfs:label>allValuesFrom</rdfs:label>
    <rdfs:domain rdf:resource="#Restriction"/>
    <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="minCardinality">
    <rdfs:label>minCardinality</rdfs:label>
    <rdfs:domain rdf:resource="#Restriction"/>
    <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</rdf:Property>
```
Properties

- `owl:ObjectProperty` and `owl:DatatypeProperty` are special cases of `rdf:Property`

```xml
<rdfs:Class rdf:ID="ObjectProperty">
  <rdfs:label>ObjectProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
</rdfs:Class>
```

Symmetric, functional and inverse functional properties can only be applied to object properties

```xml
<rdfs:Class rdf:ID="TransitiveProperty">
  <rdfs:label>TransitiveProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
</rdfs:Class>
```

- `owl:inverseOf` relates two object properties

```xml
<rdf:Property rdf:ID="inverseOf">
  <rdfs:label>inverseOf</rdfs:label>
  <rdfs:domain rdf:resource="#ObjectProperty"/>
  <rdfs:range rdf:resource="#ObjectProperty"/>
</rdf:Property>
```

Outline

1. A bit of history
2. Basic Ideas of OWL
3. The OWL Language
4. Examples
5. The OWL Namespace
6. Future Extensions
**Future Extensions of OWL**

- Modules and Imports
- Defaults
- Closed World Assumption
- Unique Names Assumption
- Procedural Attachments
- Rules for Property Chaining

**Modules and Imports**

- The importing facility of OWL is very trivial:
  - It only allows importing of an entire ontology, not parts of it
- Modules in programming languages based on information hiding: state functionality, hide implementation details
  - Open question how to define appropriate module mechanism for Web ontology languages

**Defaults**

- Many practical knowledge representation systems allow inherited values to be overridden by more specific classes in the hierarchy
  - treat inherited values as defaults
- No consensus has been reached on the right formalization for the nonmonotonic behaviour of default values

**Closed World Assumption**

- OWL currently adopts the open-world assumption:
  - A statement cannot be assumed true on the basis of a failure to prove it
  - On the huge and only partially knowable WWW, this is a correct assumption
- Closed-world assumption: a statement is true when its negation cannot be proved
  - tied to the notion of defaults, leads to nonmonotonic behaviour
Unique Names Assumption

- Typical database applications assume that individuals with different names are indeed different individuals
- OWL follows the usual logical paradigm where this is not the case
  - Plausible on the WWW
- One may want to indicate portions of the ontology for which the assumption does or does not hold

Procedural Attachments

- A common concept in knowledge representation is to define the meaning of a term by attaching a piece of code to be executed for computing the meaning of the term
  - Not through explicit definitions in the language
- Although widely used, this concept does not lend itself very well to integration in a system with a formal semantics, and it has not been included in OWL

Rules for Property Chaining

- OWL does not allow the composition of properties for reasons of decidability
- In many applications this is a useful operation
- One may want to define properties as general rules (Horn or otherwise) over other properties
- Integration of rule-based knowledge representation and DL-style knowledge representation is currently an active area of research

OWL 1.1
Conclusions

- OWL is the proposed standard for Web ontologies
- OWL builds upon RDF and RDF Schema:
  - (XML-based) RDF syntax is used
  - Instances are defined using RDF descriptions
  - Most RDFS modeling primitives are used
- Formal semantics and reasoning support is provided through the mapping of OWL on logics
  - Predicate logic and description logics have been used for this purpose
- While OWL is sufficiently rich to be used in practice, extensions are in the making
  - They will provide further logical features, including rules