OWL – Web Ontology Language

- OWL ist eine Ontologie-Beschreibungssprache
    - OWL hat ein über RDF-Schema hinausgehendes Vokabular
    - Die Syntax von OWL ist RDF
- OWL Konstrukte sind definiert in
    - http://www.w3.org/2002/07/owl
- Konstrukte von OWL:

equivalentClass
equivalentProperty
sameIndividualAs
differentFrom
allDifferent
disjointWith

unionOf
intersectionOf
complementOf

inverseOf
TransitiveProperty
SymmetricProperty
FunctionalProperty
InverseFunctionalProperty

allValuesFrom
someValuesFrom
hasValue
minCardinality
maxCardinality
cardinality

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

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
- 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

Beispiel für Reasoning (Class Membership)

- Eine Ontologie über Pizza enthält folgende Informationen
    - Mozzarella und Gorgonzola sind Käsesorten
    - Käse ist kein Fleisch und kein Fisch
    - eine vegetarische Pizza ist eine Pizza, die weder Fisch noch Fleisch als Auflage hat
- Diese Information erlaubt es, dass der Ausdruck
    - „Pizza mit (nur) Mozzarella und Gorgonzola“

  eindeutig als Spezialisierung des Ausdruck
    - „vegetarische Pizza“

  interpretiert werden kann

Quelle: Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language

---

Open 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

No Unique-Names Assumption

- OWL does not adopt the unique-names assumption of database systems
  - If 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 two staff members
  - An OWL reasoner does not flag an error
  - Instead it infers that the two resources are equal

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 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
- 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”)

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

---

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'sWeb 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 is undecidable
  - No complete (or efficient) reasoning support

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

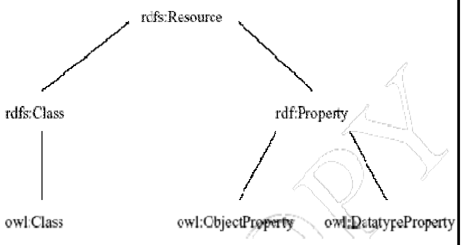
- An even further restriction limits OWL DL to a subset of the language constructors
  - E.g., OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality.
- The advantage of this is a language that is easier to
  - grasp, for users
  - implement, for tool builders
- The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

Upward Compatibility between 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



- 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

```
<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/ XLMSchema#">
```

- An OWL ontology may start with a collection of assertions for housekeeping purposes using **owl:Ontology** element

owl:Ontology

```
<owl:Ontology rdf:about="">
    <rdfs:comment>An 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** is a transitive property

Classes

- Classes are defined using **owl:Class**
    - **owl:Class** is a subclass of **rdfs:Class**
- Disjointness is defined using **owl:disjointWith**

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

Classes (2)

- **owl:equivalentClass** defines equivalence of classes

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

- **owl:Thing** is the most general class, which contains everything
- **owl:Nothing** is the empty class

Properties

- **In OWL there are two kinds of properties**
  - **Object properties, which relate objects to other objects**
    - **E.g. is-TaughtBy, supervises**
  - **Data type properties, which relate objects to datatype values**
    - **E.g. phone, title, age, etc.**

Datatype Properties

- OWL makes use of XML Schema data types, using the layered architecture of the SW

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

Object Properties

- 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>
```

Inverse Properties

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

Equivalent Properties

owl:equivalentProperty

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

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

Property Restrictions (2)

- A (restriction) class is achieved through an **owl:Restriction** element
- 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

```
<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

```
<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>
```

owl:someValuesFrom

```
<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

- We can specify minimum and maximum number using **owl:minCardinality** and **owl:maxCardinality**
- It is possible to specify a precise number by using the same minimum and maximum number
- For convenience, OWL offers also **owl:cardinality**

Cardinality Restrictions (2)

```
<owl:Class rdf:about="#course">
    <rdfs:subClassOf>
        <owl:Restriction>
                <owl:onProperty rdf:resource="#isTaughtBy"/>
                <owl:minCardinality rdf:datatype=
                "&xsd;nonNegativeInteger">
                1
                </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

Special Properties (2)

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

Boolean Combinations

- We can combine classes using Boolean operations (union, intersection, complement)

```
<owl:Class rdf:about="#course">
    <rdfs:subClassOf>
        <owl:Restriction>
                <owl:complementOf rdf:resource=
        "#staffMember"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Boolean Combinations (2)

```
<owl:Class rdf:ID="peopleAtUni">
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#staffMember"/>
        <owl:Class rdf:about="#student"/>
    </owl:unionOf>
</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

Boolean Combinations (3)

```
<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

```
<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

```
<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:

```
<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>
```

Distinct Objects

- OWL does not have the Unique Name Assumption
- To ensure that different individuals are indeed recognized as such, we must explicitly assert their inequality:

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

Distinct Objects (2)

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

```
<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

---

Versioning Information (2)

- **owl:backwardCompatibleWith** contains a reference to another ontology
    - All identifiers from the previous version have the same intended interpretations in the new version
    - Thus documents can be safely changed to commit to the new version
- **owl:incompatibleWith** indicates that the containing ontology is a later version of the referenced ontology but is not backward compatible with it

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 (2)

- 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
- Restricted anonymous classes: Anonymous classes are only allowed to occur as:
  - 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

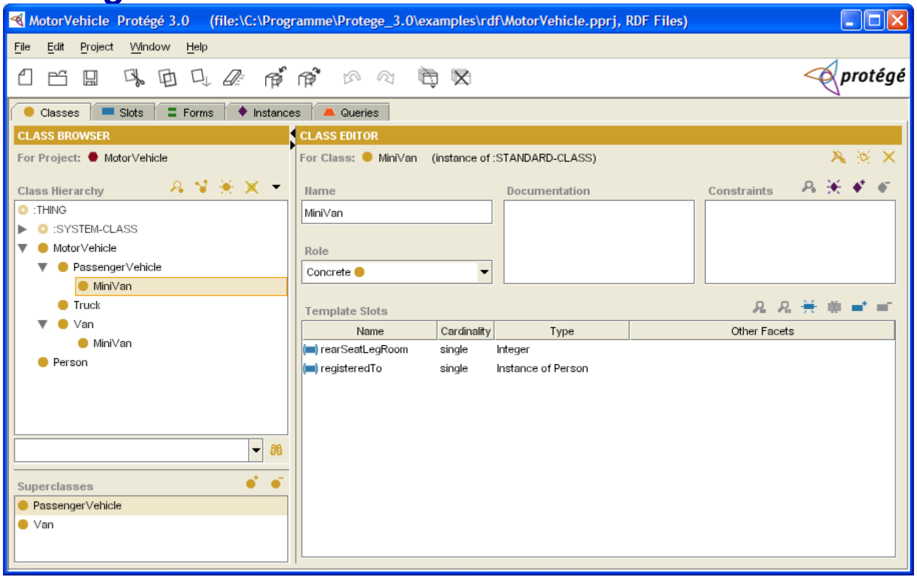
- Range restriction: **Courses must be taught by academic staff members only**
- Michael Maher is a professor
- He inherits the ability to teach from the class of academic staff members
- This is done in RDF Schema by fixing the semantics of “is a subclass of”
  - It is not up to an application (RDF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

---

Summary

- 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

Protégé



MotorVehicle Protégé 3.0 (file:\C:\Programme\Protege_3.0\examples\rdf\MotorVehicle.pprj, RDF Files)

For Project: MotorVehicle

- :THING
  - :SYSTEM-CLASS
  - MotorVehicle
    - PassengerVehicle
      - MiniVan
    - Truck
    - Van
      - MiniVan
  - Person

For Class: MiniVan (instance of :STANDARD-CLASS)

Name: MiniVan

Role: Concrete

Name	Cardinality	Type	Other Facets
rearSeatLegRoom	single	Integer	
registeredTo	single	Instance of Person	

Superclasses:
PassengerVehicle
Van