OWL

Fundamental ideas:

- OWL is proposed as a response to the need for a new language of ontolgy modeling for the Semantic Web, since he expresiveness of RDF and RDF Schema (RDFS) is much limited

Shortcomings of RDF

- RDF is mostly limited to the expression of predicates valuated as *true/false*
- RDFS is limited to the description of a hierarchy of subclasses and properties

What are OWL ontologies?

Ontologies are used to capture knowledge

- Describes concepts in some domain of interest (*domain of discurse*)
- Describes the relationships that hold between those concepts

OWL characteristics

- With a rich set of operators: *intersection*, *union*, *negation*
- Based on a logical model that allows concept definition as well as their description
- Complex concepts built up in definitions from simpler ones
- Uses a *reasoner*

Requirements of a language for ontology description

It must provide us with an efficient *reasoning support* and make easy the expression of complex properties

RDF Schema + Complete Logic:

It does give the previous requirement but the language is so powerful that becomes *undecidable* for a computer program

Solution proposed by W3C Web Ontology Working Group:

To define three languages for ontology description:

- OWL Full
- OWL DL
- OWL Lite

OWL Full

Language chaaracteristics:

- It uses all the primitives of OWL and allows arbitrary mixing of these primitives with RDF and RDFS constructs within an ontology specification
- It allows to apply language's primitives each other
- It can be imposed a restriction on the *class of all classes*, e.g., for limiting the number of an ontology classes
- It becomes a language so powerful that it turns out to be undecidable

OWL DL

OWL Description Logic

- This language put constraints on the way of using mathematical constructors of OWL Full and RDF
- It does not allow that OWL constructors are applied to each other
- The resulting language conforms to a well-defined *description logic* (DL)
- It allows us to obtain efficient reasoning-suport tools
- However, the compatibility with RDF is lost

OWL Lite

It is an even more restrictive subset of OWL than OWL-DL

- It further restricts OWL-DL constructors on classes, i.e.: OWL Lite does not allow enumerated types, or disjunction sentences, or an arbitrary cardinality of classes
- Easy to learn by its users
- Easy to implement in the development of software tools
- Its expresiveness is very restrictive

OWL sublanguages compatibility

Developed ontologies:

OWL Lite $\sqsubseteq$ OWL DL $\sqsubseteq$ OWL Full

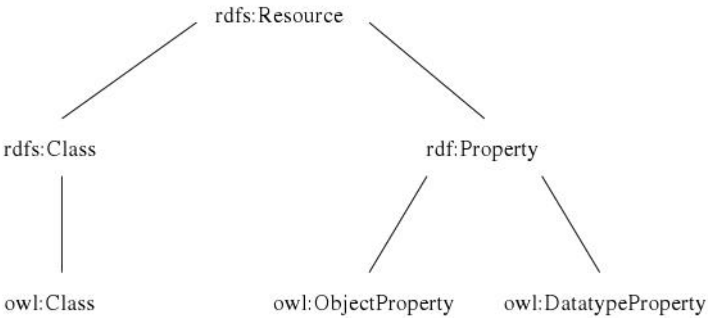
Upwards compatibility:

Conclusions with OWL Lite $\sqsubseteq$ OWL DL $\sqsubseteq$ OWL Full

Syntantical compatibility:

- All the OWL language variants use the RDF syntax
- The *instances* are declared as in RDF, i.e., by using RDF description and type information

Relationship between OWL subclasses and RDF/RDFS



OWL ontologies -II

Components of OWL ontologies

- Individuals
- Properties
- Classes

OWL ontologies components

Individuals

- Represent objects in the *domain of discourse*
- Unique Name Assumption (UNA) is not assumed (differently from Protégé)
- Individuals are also known as being *instances of classes*

OWL ontologies components

Properties

- Are understood as *binary relations* (relations between 2 “things”) on individuals
- Inverse of a property
- Types of properties:
  - *functional*: they have only 1 value
  - *transitive*
  - *symmetric*
- Properties are equivalent to *slots* in Protégé

OWL ontologies components

Classes

- Are *sets* that contain individuals
- In OWL classes are built up from descriptions
- Mathematically described to state precisely the requirements for individuals membership
- Organised in subclass/superclass hierarchies or *taxonomies*
- Subsumption (*is a super/sub class of*) relationships can be automatically computed by the reasoner in OWL-DL
- *Concept* is used in place of *class* since classes are a concrete representation of concepts

The usefulness of *formal semantics*

Fundamental idea:

To let people be able to reason about the acquired/stored *knowledge*

Examples of Formal Semantics-based reasoning:

- Class membership
- Class equivalence
- Consistence tests:
  To detect inconsistencies in the following expression,
  $x \in A;\ A \subseteq B \cap C;\ A \subseteq D \ \wedge B \cap D = \{0\}$
- Classification:
  Should a property (*property*, *value*) be class-membership sufficient condition and should an element satisfy that condition, it has to be an instance of the class

The usefulness of *formal semantics* –II

Fundamental idea:

- Semantics is thus a pre-requisite for *supporting reasoning*

Supporting reasoning:

- To check the consistency of ontologies
- To detect non desired relationships between classes
- Automatic classification of class instances

The so-called “description logics” (DL) must be understood as a subset of the Predicate Logic for which counting with an efficient reasoning support becomes then possible

OWL reasoner

Characteristics

- To check that all the statements and definitions in a given ontology are mutually consistent
- To identify which concepts fit under which definitions
- To maintain the class hierarchy correctly (especially, when classes have more than one parent)

OWL Syntax

The root element of OWL is also a member of `rdf:RDF`

```
<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#">
```

An OWL code can start with a collection of assertions, useful to organise the document:

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

Class elements

```
<owl:Class rdf:ID="AssociateProfessor">
  <rdfs:subClassOf rdf:resource="#AcademicStaffMember">
  </rdfs:subClassOf>
</owl:Class>
```

OWL classes are assumed to *overlap* if not specified otherwise

- We cannot assume that an individual cannot be member of a particular class
- To make an individual member of only one class in a group of classes we have to make each class disjoint from one another

Disjoint classes

Logical disjunction expression (separation of a group of classes regarding individuals' membership):

```
<owl:Class rdf:about="#AssociateProfessor">
  <owl:disjointWith rdf:resource="#Professor">
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#AssistantProfessor">
  </owl:disjointWith>
</owl:Class>
```

`equivalentClass` operation

Built-in property that links a class description to another class description:

- A class axiom may contain (multiple) `owl:equivalentClass` statements
- The two class descriptions involved have the same class extension (contain the same set of individuals)
- But it does not imply class equality, i.e., both classes need not to denote the same concept

```
<owl:Class rdf:ID="Faculty">
  <owl:equivalentClass rdf:resource="#AcademicStaffMember">
  </owl:equivalentClass>
</owl:Class>
```

OWL Properties

Property in OWL

- Represents relationships between individuals or between an individual and data values
- Types of properties:
  - Object properties
  - Datatype properties
  - Annotation properties

Property elements in OWL

creation of an *object property*:

- Object properties examples: `isTaughtBy, supervises`

Objects' property:

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

Subproperties

- Specialise their *super properties*
- It is possible to form hierarchies of properties in OWL

Property elements in OWL - II

creation of a *datatype property*:

- Datatypes properties examples: `telephone, title, age`

Data Types' property:

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

Subproperties

- It is possible to create subproperties of datatype properties
- It is not possible to mix and match object (sub)properties with datatype (sub)properties

Property domains and ranges

Fundamental idea

Properties link individuals from the *domain* to individuals from the *range*

OWL particularity regarding domains

- OWL domains and ranges cannot be seen as constraints to be checked
- They are only used as reasoning *axioms*, i.e., `donkeyBobo is taughtBy professorPeter` wouldn't generally result in error! (it could even be used by the reasoner to infer the class Donkey as a subclass of Course!)

Property domains and ranges-II

- It is possible to specify multiple classes as the range for a property
- In some frameworks (like Protégé) the range of the property is interpreted to be the intersection of the classes
- Any individuals that are used on the *right hand side* of a property will be inferred to be members of the *range* class
- Viceversa with individuals on the *left hand side* and the domain of a property
- In the two cases above the individuals may have not been asserted to be member of the class and then it's inferred so

Inverse properties

Fundamental idea

Each object property described in OWL may have a corresponding inverse property

Specification/creation

Frameworks like Protégé offer an *inverse property view* to their users for mapping inverse properties:

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

Inverse properties-II

Domain and ranges of a property and the inverse property are swapped out

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

Property equivalence:

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

OWL Object Property Characteristics

Miscelanea

- Functional properties are also known as *single valued properties* and also *features*
- If a property is *inverse functional* then it means that its *inverse* property is also *functional*
- If a property is *transitive*, it cannot be functional
- The inverse of transitive property should also be transitive
- A symmetric property coincides with its own inverse property
- OWL-DL does not allow Datatype properties to be transitive, symmetric or have inverse properties

Expression of special properties

Directly defined

- owl:TransitiveProperty
- owl:SymmetricProperty
- owl:FunctionalProperty: this property can only have a value
- owl:InverseFunctionalProperty: defines an injection (2 different elements have $\neq$ value)

```
<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>
```

Datatype properties in OWL

Use of these properties

- To state that all the individuals of an anonymous class have (*at least one* or *only those*) specific type value
- We can also specify restrictions on the possible values of the individuals, e.: `hasCalorificValue some integer [>=400]`
- Then, we can use the *reasoner* to perform instance classification
- The only type of property accepted is *functional*

Restrictions on properties

Fundamental idea

A class of individuals is defined by the relationships that these individuals participate in

Class description in OWL

- It uses *restrictions* to do relationship-based class definitions
- A restriction is a *kind of* class, as a named class is a kind of class too
- Restriction categories:
  - Quantifier restrictions: *Existential*($\exists$), *Universal*($\forall$)
  - Cardinality restrictions
  - `hasValue` restrictions

Existential restrictions

Fundamental idea

Class of individuals that have *at least one* (or *some*) relationship along a specified property to individuals that are members of another class

Also known as *some* or *some values from* restriction :

```
<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>
```

Restrictions on properties-II

Fundamental idea

Existential relationships do not mandate that the only mapping for the given property must be to the individuals of the *filler* class, i.e., an `AcademicStaffMember` can teach a graduate course too!

Restrictions on properties-III

Universal restriction

- Needed to restrict the relationships for a given property to the individuals that are members of a specific filler class
- Universal restrictions do not specify the existence of a relationship

```
<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>
```

Any course that is not given this year will satisfy the restriction above!

Necessary and sufficient conditions

Fundamentals

- With necessary conditions (those that are established by `someValuesFrom` restrictions) we cannot say that any (random) individual that satisfies these conditions must be a member of the *class* being defined
- A class that only has *necessary conditions* is known as a "Primitive Class"
- Necessary conditions are called *Superclasses* in Protégé
- A class that has sufficient and necessary conditions is a "Defined Class"

Restrictions on properties-IV

hasValue ($\in$) restriction definition

Describes an anonymous class of *individuals* that are related to another *specific individual* along a specified property

Fundamentals

- Different from the quantifier *some* restriction, though both seem to be quite similar: $\exists$ describes individuals related to any individual of the filler class
- There could be other relationships along the specified property
- `hasValue` restrictions are semantically equivalent to an $\exists$ restriction which has a filler class that contains only one individual

`hasValue` restrictions

Restriction on the values of one property: `hasValue`:

```
<owl:Class rdf:about="#MathCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:hasValue rdf:resource="#949318"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Restrictions on properties-V

Use of cardinality $=$ hasProperty

To specify the exact number of relationships that an individual must participate in for a given property

$\geq$ hasProperty **n**

- Describes the individuals of an anonymous class that participate in *at least n Property* relationships

$\leq$ hasProperty **n**

- Describes the individuals of an anonymous class that participate in *at most n Property* relationships
- Places no minimum limit on the number of relationships

Restrictions on properties-V

Cardinality restrictions:

```
<owl:Class rdf:about="#Department">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember"/>
        <owl:minCardinality rdf:datatype="&xsd;
            nonNegativeInteger"> 3
        </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#hasMember"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger"
         > 30
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Boolean combinations of classes

We can define operations on the classes: union, intersection, complement

course is an instance of the complement of `StaffMember`: (`Course` and `AcademicStaffMember` are *disjoint* classes in Protégé:

```
<owl:Class rdf:about="#Course">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:complementOf rdf:resource="#StaffMember"/>
     </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Or a union of classes:

```
<owl:Class rdf:ID="PeopleAtUni">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#StaffMember"/>
    <owl:Class rdf:about="#Student"/>
  </owl:unionOf>
</owl:Class>
```

Boolean combinations of classes-II

Boolean combinations can be arbitrarily nested:

```
<owl:Class rdf:ID="AdminStaff">
  <owl:intersectionOf rdf:parseType="Collection">
   <owl:Class rdf:about="#AcademicStaffMember"/>
   <owl:Class>
    <owl:complementOf>
     <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Faculty"/>
        <owl:Class rdf:about="#TechSupportStaff"/>
        </owl:unionOf>
      </owl:Class>
    </owl:complementOf>
  </owl:Class>
 </owl:intersectionOf>
</owl:Class>
```

Enumerations

An enumeration is an element: `owl:oneOf`, used for the definition of one class by listing all its elements:

```
<owl:Class rdf:ID="#AcademicStaffCode">
  <owl:oneOf rdf:parseType="Collection">
   <owl:Thing rdf:about="#947744"/>
   <owl:Thing rdf:about="#949111"/>
   <owl:Thing rdf:about="#949318"/>
   <owl:Thing rdf:about="#949352"/>
   ...
   <owl:Thing rdf:about="#959312"/>
   <owl:Thing rdf:about="#961111"/>
   <owl:Thing rdf:about="#986677"/>
  </owl:oneOf>
</owl:Class>
```

Instances

Fundamental ideas:

- Instances are declared as in RDF or in an abbreviated form: `<academicStaffMember rdf:ID="949352"/>`
- OWL does not assume the *name unicity* convention used by database systems, i.e., 2 instances with different ID do not have to be different individuals
- OWL has operators to indicate explicitly inequalities between individuals: `owl:DifferentFrom` y `owl:AllDifferent`

Instance declaration examples

Each course is taught, at most, by one teacher:

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>
```

Writing the text below does not yield an OWL reasoner error:

```
<course rdf:ID="CIT1111">
  <isTaughtBy rdf:resource="#949318"/>
  <isTaughtBy rdf:resource="#949352"/>
</course>
```

Distinct professor-IDs represent different professors:

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
   <AcademicStaffCode rdf:about="#949318"></AcademicStaffCode>
   <AcademicStaffCode rdf:about="#949352"></AcademicStaffCode>
   <AcademicStaffCode rdf:about="#949111"></AcademicStaffCode>
  </owl:distinctMembers>
</owl:AllDifferent>
```

Example: printer ontology

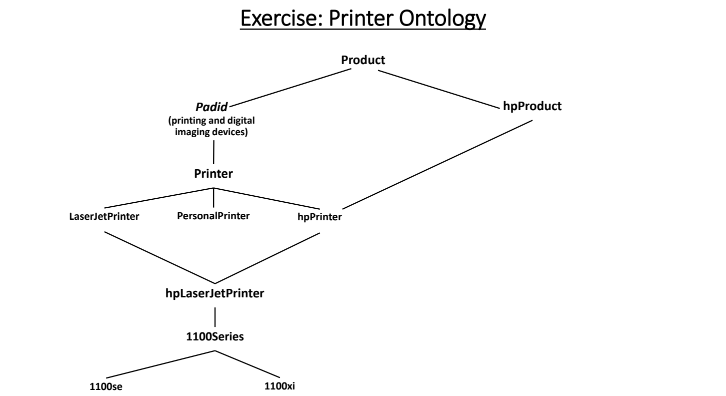


Figure: Example shown in: “Cooperative Information Systems : Semantic Web Primer”. *Antoniou, Grigoris, Van Harmelen, Frank*, MIT Press(2008)

Example: printer ontology-2

RDF/OWL Header:

```
<!DOCTYPE owl
   [ <!ENTITY xsd ]> "http://www.w3.org/2001/XMLSchema#" >
<rdf:RDF 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#"
   xmlns:owl ="http://www.w3.org/2002/07/owl#"
   xmlns="http://www.cs.vu.nl/~frankh/spool/printer.owl#">
```

Beginning of OWL document:

```
<owl:Ontology rdf:about="">
  <owl:versionInfo> Mi ejemplo version 1.3, 17 Enero 2017
   </owl:versionInfo>
 </owl:Ontology>
```

Example: printer ontology–3

```
<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 imgaging devices are a
       subclass of product.
   </rdfs:comment>
  <rdfs:label>Device</rdfs:label>
  <rdfs:subClassOf rdf:resource="#product">
   </rdfs:subClassOf>
</owl:Class>
```

Example: printer ontology–4

```
<owl:Class rdf:ID="hpProduct">
 <rdfs:comment>
  HP products are exactly those products manufactured by Hewlett
      Packard.
 </rdfs:comment>
 <owl:intersectionOf rdf:parseType="Collection">
   <owl:Class rdf:about="#product"></owl:Class>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#manufactured_by">
     </owl:onProperty>
     <owl:hasValue rdf:datatype="&xsd;string">
       Hewlett Packard
     </owl:hasValue>
   </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

Ex

```
<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>
```

```
<owl:Class rdf:ID="hpPrinter">
  <rdfs:comment>
    HP printers are  HP Products and printers.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#printer"/>
  <rdfs:subClassOf rdf:resource="#hpProduct"/>
</owl:Class>
```

Example: printer ontology–6

```
<owl:Class rdf:ID="laserJetPrinter">
  <rdfs:comment>
  Laser jet printers are exactly those printers that use laser
      jet printing technology.
  </rdfs:comment>
  <owl:intersectionOf rdf:parseType="Collection">
   <owl:Class rdf:about="#printer">
   <owl:Restriction>
     <owl:onProperty rdf:resource="#printingTechnology">
     </owl:onProperty>
     <owl:hasValue rdf:datatype="&xsd;string">
      laser jet
     </owl:hasValue>
   </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Example: printer ontology–7

```
<owl:Class rdf:ID="hpLaserJetPrinter">
  <rdfs:comment>
   HP laser jet printers are HP products and laser jet printers.
    </rdfs:comment>
   <rdfs:subClassOf rdf:resource=#laserJetPrinter"/>
   <rdfs:subClassOf rdf:resource=#hpPrinter"/>
 </owl:Class>
```

```xml
<owl:Class rdf:ID="1100series">
  <rdfs:comment>
  1100series printers are 8ppm printing speed and HP laser jet
      printers with 600dpi printing resolution.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#hpLaserJetPrinter"/>
  <rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#printingSpeed">
    </owl:onProperty>
    <owl:hasValue rdf:datatype="&xsd;string"> 8ppm
    </owl:hasValue>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#printingResolution">
    </owl:onProperty>
    <owl:hasValue rdf:datatype="&xsd;string"> 600dpi
    </owl:hasValue>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Example: printer ontology–9

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

Example: printer ontology–10

```
<owl:Class rdf:ID="1100xi">
  <rdfs:comment>
    1100xi printers belong to the 1100 series and cost 350 USD.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#1100series">
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price">
      </owl:onProperty>
      <owl:hasValue rdf:datatype="&xsd;integer"> 350
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Example: printer ontology–11

```
<owl:DatatypeProperty rdf:ID="manufactured_by">
 <rdfs:domain rdf:resource="#product"></rdfs:domain>
 <rdfs:range rdf:resource="&xsd;string"></rdfs:range>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="price">
 <rdfs:domain rdf:resource="#product"></rdfs:domain>
 <rdfs:range rdf:resource="&xsd;nonNegativeInteger"></rdfs:range
     >
</owl:DatatypeProperty>
```

Example: printer ontology–12

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