Chapter 10

DEVELOPING AN OWL ONTOLOGY FOR E-TOURISM

Jorge Cardoso
Department of Mathematics and Engineering, University of Madeira, 9000-390, Funchal, Portugal – jcardoso@uma.pt

1. INTRODUCTION

Currently, the World Wide Web is mainly composed of documents written in Hyper Text Markup Language (HTML). HTML is a language that is useful for visual presentation and for direct human processing (reading, searching, browsing, querying, filling in forms, etc). HTML documents are often handwritten or machine generated and often active HTML pages. Most of the information on the Web is designed only for human consumption. Humans can read HTML documents and understand them, but their inherent meaning is not shown to allow their interpretation by computers.

To surpass this limitation, the W3C (World Wide Web Consortium, www.w3.org) has been working on approaches to define the information on the Web in a way that it can be used by computers not only for display purposes, but also for automation, interoperability, and integration between systems and applications. One way to enable machine-to-machine understanding, exchange, and automated processing is to make Web resources more readily accessible by adding meta-data annotations that describe their content in such a way that computers can understand it. This is precisely the objective of the semantic Web – to make the information on the Web understandable and useful to computer applications in addition to humans. "The semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation." (Berners-Lee, Hendler et al. 2001).

The W3C has proposed a language designed for publishing and sharing data, and automating data understanding by computers using ontologies on the Web. The language, called OWL (Web Ontology Language), will transform the current Web to the concept of Semantic Web. OWL is being planned and designed to provide a language that can be used for applications that need to understand the meaning of information instead of just parsing data for display purposes.

2. OWL AND THE SEMANTIC WEB STACK

The semantic Web identifies a set of technologies, tools, and standards which form the basic building blocks of an infrastructure to support the vision of the Web associated with meaning. The semantic Web architecture is composed of a series of standards organized into a structure that is an expression of their interrelationships. This architecture is often represented using a diagram first proposed by Tim Berners-Lee (Berners-Lee, Hendler et al. 2001). Figure 10-1 illustrates the different parts of the semantic Web architecture. It starts with the foundation of URIs and Unicode. On top of that we can find the syntactic interoperability layer in the form of XML, which in turn underlies RDF and RDF Schema (RDFS). Web ontology languages are built on top of RDF and RDFS. The last three layers are logic, proof, and trust, which have not been significantly explored. Some of the layers rely on the digital signature component to ensure security.

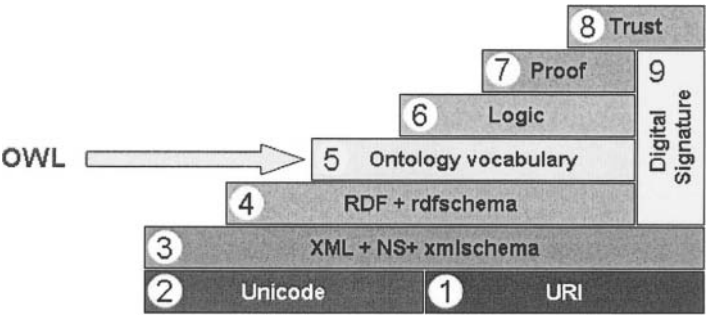

Figure 10-1. Semantic Web layered architecture (Berners-Lee, Hendler et al. 2001)

In the following sections we briefly describe these layers. While the notions presented have been simplified, they give a reasonable conceptualization of the various components of the semantic Web.

2.1 URI and Unicode

A Universal Resource Identifier (URI) is a formatted string that serves as a way for identifying abstract or physical resource. Uniform Resource Locator (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism. A Uniform Resource Name (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable. For example,

- The URL http://dme.uma.pt/jcardoso/index.htm identifies the location where a Web page can be retrieved from
- The URN urn:isbn:3-540-24328-3 identifies a book using its ISBN

Unicode provides a unique number for every character, independently of the underlying platform, program, or language. Before the creation of unicode, there were various different encoding systems that made the manipulation of data too complex. Any given computer needed to support many different encodings. There was always the risk of encoding conflict, since two encodings could use the same number for two different characters, or use different numbers for the same character.

2.2 XML

XML is accepted as a standard for data interchange on the Web allowing the structuring of data but without communicating its meaning. It is a language for semi-structured data and has been proposed as a solution to solve integration problems, because it allows a flexible coding and display of data.

While XML has gained much of the world's attention it is important to recognize that XML is simply a way to standardize data formats. But, from the point of view of semantic interoperability, XML has limitations. One significant aspect is that there is no way to recognize the semantics from a particular domain because XML aims at document structure and imposes no common interpretation of the data (Decker, Melnik et al. 2000). Another problem is that XML has a weak data model incapable of capturing relationships or constraints. While it is possible to extend XML to incorporate rich metadata, XML does not allow supporting automated interoperability of systems without human involvement. Even though XML is simply a data-format standard, it is part of the set of technologies that constitute the foundations of the semantic Web.

2.3 RDF

On the top of XML, the W3C has developed the Resource Description Framework (RDF) (RDF 2002) language to standardize the definition and use of metadata. Therefore, XML and RDF each have their merits as a foundation for the semantic Web, but RDF provides more suitable mechanisms for developing ontology representation languages like OIL (Horrocks, Harmelen et al. 2001) or OWL (OWL 2004).

RDF uses XML and it is at the base of the semantic Web, so that all the other languages corresponding to the upper layers are built on top of it. RDF is a formal data model for machine understandable metadata used to provide standard descriptions of Web resources. By providing a standard way of referring to metadata elements, specific metadata element names, and actual metadata content, RDF builds standards for applications so that they can interoperate and intercommunicate more easily, facilitating data and system integration and interoperability. In a first approach it may seen that RDF is very similar to XML, but a closer analysis reveals that they are conceptually different. If we model the information present in a RDF model using XML, human readers would probably be able to infer the underlying semantic structure, but applications would not.

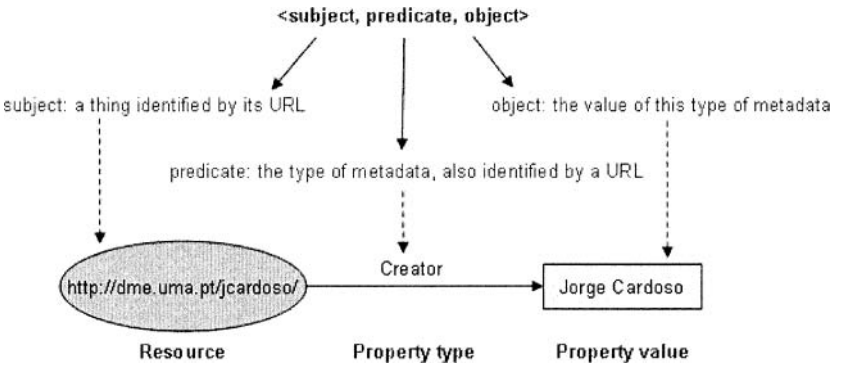

Figure 10-2. An RDF statement

RDF is a simple general purpose metadata language for representing information in the Web and provides a model for describing and creating relationships between resources. A resource can be a thing, such as a person, a song, or a Web page. With RDF it is possible to add pre-defined modeling primitives for expressing semantics of data to a document without making any assumptions about the structure of the document. RDF defines a resource as any object that is uniquely identifiable by a URI (Universal

Resource Identifier). Resources have properties associated to them. Properties are identified by property-types, and property-types have corresponding values. Property-types express the relationships of values associated with resources. The basic structure of RDF is very simple and basically uses RDF triples of the form <subject, predicate, object> as illustrated in Figure 10-2.

2.4 RDF Schema

The RDF Schema (RDFS 2004) provides a type system for RDF. The RDFS is technologically advanced compared to RDF since it provides a way to build an object model from which the actual data is referenced and which tells what things really mean.

Briefly, the RDF Schema (RDFS) allows users to define resources with classes, properties, and values. The concept of RDF class is similar to the concept of class in object-oriented programming languages such as Java and C++. A class is a structure of similar things and inheritance is allowed. This allows resources to be defined as instances of classes and subclasses of classes allowing classes to be organized in a hierarchical fashion. For example, the class First_Line_Manager might be defined as a subclass of Manager which is a subclass of Staff, meaning that any resource which is in class Staff is also implicitly in class First_Line_Manager as well.

An RDFS property can be viewed as an attribute of a class. RDFS properties may inherit from other properties, and domain and range constraints can be applied to focus their use. For example, a domain constraint is used to limit what class or classes a specific property may have and a range constraint is used to limit its possible values. With these extensions, RDFS comes closer to existing ontology languages. As with RDF, the XML namespace mechanism serves to identify RDFS.

2.5 Ontologies

An ontology is an agreed vocabulary that provides a set of well-founded constructs to build meaningful higher level knowledge for specifying the semantics of terminology systems in a well defined and unambiguous manner. For a particular domain, an ontology represents a richer language for providing complex constraints on the types of resources and their properties. Compared to a taxonomy, ontologies enhances the semantics by providing richer relationships between the terms of a vocabulary. Ontologies are usually expressed in a logic-based language, so that detailed and meaningful distinctions can be made among the classes, properties, and relations.

Ontologies can be used to increase communication both between humans and computers. The three major uses of ontologies (Jasper and Uschold 1999) are:

- To assist in communication between humans.
- To achieve interoperability and communication among software systems.
- To improve the design and the quality of software systems.

Currently, the most prominent ontology language is OWL (OWL 2004), the language we will cover in this chapter. OWL is a vocabulary extension of RDF and is derived from the DAML+OIL language (DAML 2001), with the objective of facilitating a better machine interpretability of Web content than the one supported by XML and RDF. This evolution of semantic Web languages is illustrated in Figure 10-3.

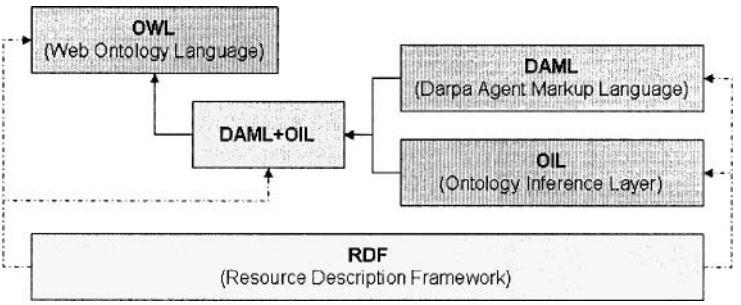

Figure 10-3. Evolution of Semantic Web Languages

DAML+OIL resulted from the integration of the DAML and OIL languages. DAML (DARPA Agent Markup Language) was created as part of a research program (www.daml.org) started in August 2000 by DARPA, a US governmental research organization. OIL (Ontology Inference Layer) is an initiative funded by the European Union programme for Information Society Technologies. OIL was intended to support e-commerce and enable knowledge management. OIL and DAML were merged originating DAML+OIL, which later evolved into OWL.

3. LIMITATIONS OF RDFS

RDF Schema is a semantic extension of RDF and it is used for describing vocabularies in RDF. It provides mechanisms for describing groups of related resources and the relationships between resources. These resources

are used to determine characteristics of other resources, such as the domains and ranges of properties.

However, RDFS is a very primitive language and a more expressive solution is advantageous to describe resources in more detail. In order to fully understand the potentialities of OWL, it is important to identify the limitations that RDFS suffers from. It is the recognition of the limitations of RDFS that led to the development of OWL.

Let's analyze some of the limitations of RDFS to identify the extensions that are needed:

1. RDFS cannot express equivalence between concepts. This is important to be able to express the equivalence of ontological concepts developed by separate working groups.
2. RDFS does not have the capability of expressing the uniqueness and the cardinality of properties. In some cases, it may be necessary to express that a particular property value may have only one value in a particular class instance. For example, a sedan car has exactly four wheels and a book is written by at least one author.
3. RDFS can express the values of a particular property but cannot express that this is a closed set by enumeration. . For example, the gender of a person should have only two values: male and female.
4. RDFS cannot express disjointedness. For example, the gender of a person can be male and female. While it is possible in RDFS to express that John is a male and Julie a female, there is no way of saying that John is not a female and Julie is not a male.
5. RDFS cannot build new classes by combining other classes using union, intersection, and complement. For example, the class "staff" might be the union of the classes "CEO", "manager" and "clerk". The class "staff" may also be described as the intersection of the classes "person" and "organization employee". Another example is the ability to express that a person is the disjoint union of the classes male and female.
6. RDFS cannot declare range restrictions that apply to some classes only. The element rdfs:range defines the range of a property for all classes. For example, for the property "eats", it is not possible to express that cows eat only plants, while other animals may eat meat, too.
7. RDFS cannot express special characteristics of properties such as transitive property (e.g. "more complex than"), unique property (e.g. "is mother of"), and that a property is the inverse of another property (e.g. "writes" and "is written by")

4. THREE TYPES OF OWL

Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. In OWL, an ontology is a set of definitions of classes and properties, and constraints on the way those classes and properties can be employed.

In the previous sections, we have established that RDFS was one of the base models for the semantic Web, but that it suffered from several limitations. At the top of the RDFS layer it is possible to define more powerful languages to describe semantics. The most prominent markup language for publishing and sharing data using ontologies on the Internet is the Web Ontology Language (OWL). OWL adds a layer of expressive power to RDFS, providing powerful mechanisms for defining complex conceptual structures, and formally describes the semantics of classes and properties using a logical formalism.

OWL has been designed to meet the need for a Web ontology language. As already mentioned, XML gives a syntax for semi-structured documents but does not associate an XML tag with semantics. Therefore, XML tags do not carry out any meaning, at least for computers. XML Schema gives a schema to XML documents and extends XML with a broad set of data types. RDF is a simple data model represented using the XML syntax for resources and the relations between them. The RDF Schema provides a type system for RDF which allows users to define resources with classes, properties, and values. It provides a vocabulary for describing properties and classes of RDF resources. The RDFS is technologically advanced compared to RDF since it provides a way to build an object model from which the actual data is referenced and which tells what things really mean. OWL goes a step further and allows for describing properties and classes, such as property type restrictions, equality, property characteristics, class intersection, and restricted cardinality.

OWL is the proposed standard for Web ontologies. It builds upon RDF and RDF Schema. XML-based RDF syntax is used, instances are defined using RDF descriptions, and most RDFS modeling primitives are also used. The W3C's Web Ontology Working Group defined OWL as three different sublanguages:

- OWL Lite
- OWL DL
- OWL Full

Each sublanguage fulfils different requirements. OWL Lite supports those users primarily needing a classification hierarchy and simple constraint features. The advantage of OWL Lite is that it is a language that is easier for users to understand and it is also easier for developers to implement tools

and applications than the more complicated and wide-ranging DL and Full versions. The main disadvantage is that it has a restricted expressivity. For example, it does not support the concept of disjunction, excludes enumerated classes, and cardinality is restricted to only 0 or 1.

OWL DL supports those users who want maximum expressiveness. OWL DL is more expressive but still ensures completeness and decidability, i.e. all the calculations will compute and terminate. OWL DL (DL for description logics) corresponds to a field of research concerning a particular fragment of decidable first order logic.

OWL Full has maximum expressivity and the syntactic freedom of RDF but does not guarantee computation. It uses all the OWL language primitives and the combination of these primitives in arbitrary ways with RDF and RDF Schema. One major problem is that OWL Full is so expressive that it is undecidable.

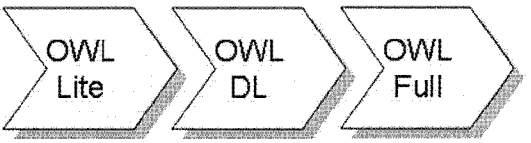

Figure 10-4. OWL sublanguages

According to Figure 10-4, every OWL Lite ontology or conclusion is a legal OWL DL ontology or conclusion, but not the inverse, and so on for OWL DL and OWL Full.

5. OWL ONTOLOGY DEVELOPMENT

Tourism is a data rich domain. Data is stored in many hundreds of data sources and many of these sources need to be used in concert during the development of tourism information systems. Our e-tourism ontology provides a way of viewing the world of tourism. It organizes tourism related information and concepts. The e-tourism ontology provides a way to achieve integration and interoperability through the use of a shared vocabulary and meanings for terms with respect to other terms.

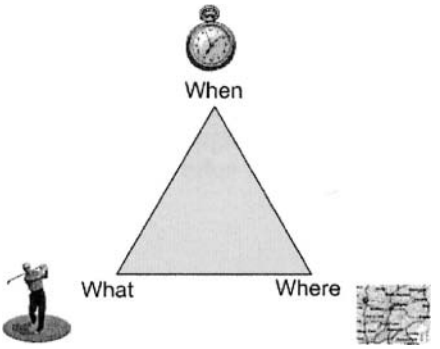

Figure 10-5. What, Where, and When

The e-tourism ontology was built to answer three main questions (Figure 10-5) that can be asked when developing tourism applications: What, Where, and When.

- **What**. What can a tourist see, visit and what can he do while staying at a tourism destination?
- **Where**. Where are the interesting places to see and visit located? Where can a tourist carry out a specific activity, such as playing golf or tennis.
- **When**. When can the tourist visit a particular place? This includes not only the day of the week and the hours of the day, but also the atmospheric conditions of the weather. Some activities cannot be undertaken if it is raining for example.

Constructing an ontology is a time-consuming task since it is necessary to find out information about real tourism activities and infrastructures and feed them into the knowledge base.

In the next section, we will be construction an OWL ontology for e-tourism. Since RDFS and OWL are compatible, the ontology developed will contain RDFS elements within the OWL syntax. For those who dislike writing ontologies by hand, a few ontology editors are available. We recommend using one of the most well-know ontology editors, Protégé, which is illustrated in Figure 10-6, to develop the ontology presented in the next section.

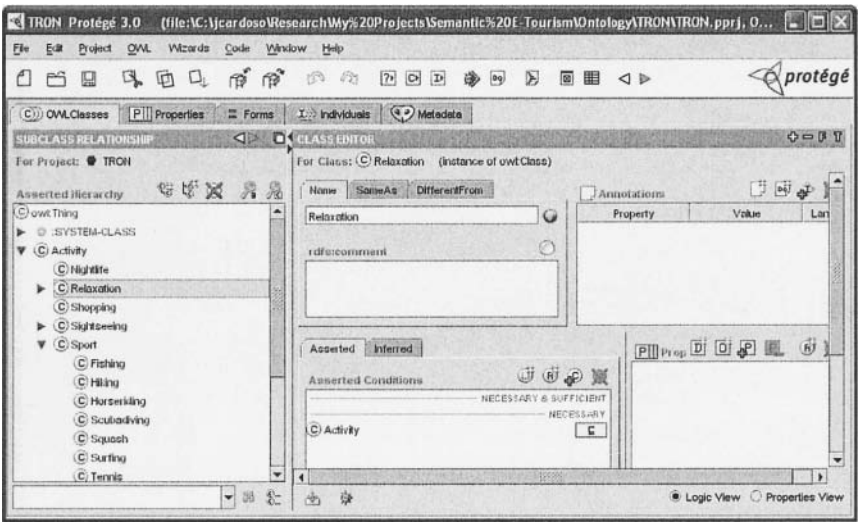

Figure 10-6. Creating the e-tourism ontology using Protégé editor

5.1 Header

An OWL ontology starts with a set of XML namespace declarations enclosed in an opening rdf:RDF tag. XML namespaces allow a means to unambiguously interpret identifiers and make the rest of the ontology presentation much more readable. A namespace is declared using three elements: the reserved XML attribute xmlns, a short prefix to identify the namespace, and the value which must be a URI (Uniform Resource Identifier) reference. An example of a namespace for our e-tourism ontology is:

```
<rdf:RDF
...
  xmlns:weather="http://dme.uma.pt/owl/weather#"
...
>
```

Our initial set of XML namespace declarations which is enclosed in an opening rdf:RDF tag is the following:

```
<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#">
  xmlns  =" http://dme.uma.pt/jcardoso/owl/e-tourism#"
  xml:base="http://dme.uma.pt/jcardoso/owl/e-
tourism#">
  xmlns:weather="http://dme.uma.pt/owl/weather#"
```

The first four namespace declarations are conventional declarations. They are used to introduce the OWL (xmlns:owl), RDF (xmlns:rdf), and RDFS (xmlns:rdfs) vocabularies, and XML Schema (xmlns:xsd) datatypes.

The following three declarations identify the namespace associated with our ontology. The first makes it the default namespace, stating that unprefixed qualified names refer to the current ontology. The second identifies the base URI for our ontology. The third declaration identifies the namespace of the supporting weather ontology with the prefix weather. The URI for an identifier is the concatenation of the xml:base value (or the document URL if there is no xml:base) with "#" and the identifier. Thus, the complete URI for an OWL class named ABC is http://dme.uma.pt/owl/e-tourism#ABC.

Once the namespaces are specified, an OWL ontology specifies a set of assertions grouped under the owl:Ontology element. The assertions include the version information which assumes that different versions of the ontology may possibly be developed. The main assertions that can be made about the versioning are:

- owl:versionInfo – a statement which generally contains a string giving information about the version of the ontology.
- owl:priorVersion – a statement that makes reference to another ontology indicating earlier versions of the current ontology. This statement can be used by ontology management tools and applications.
- owl:backwardCompatibleWith – contains a reference to another ontology and indicates that all identifiers from the previous version have the same intended interpretations in the new version.
- owl:incompatibleWith – a statement contains a reference to another ontology indicating that the ontology is a newest version of the referenced ontology but is not backward compatible with it.
- owl:imports – provides support for integrating definitions specified in another OWL ontology published on the Web and identified by a URI. The meaning of the imported ontology is considered to be part of the meaning of the importing ontology.

For example:

```
<rdf:RDF
...
<owl:Ontology rdf:about="">
  <rdfs:comment>E-Tourism OWL Ontology
  </rdfs:comment>
  <owl:versionInfo> v.1 2005-10-25
  </owl:versionInfo>
  <owl:priorVersion>
    <owl:Ontology rdf:about=
  "http://dme.uma.pt/jcardoso/owl/tourism.owl"/>
  </owl:priorVersion>
  <owl:backwardCompatibleWith
      rdf:resource="http://dme.uma.pt/owl/tourism"/>
  <owl:imports
        rdf:resource="http://math.uma.pt/owl/places"/>
  <rdfs:label>E-Tourism Ontology</rdfs:label>
...
</owl:Ontology>
...
</rdf:RDF>
```

Between the header and the closing rdf:RDF tag is the definition of the ontology itself.

5.2 Classes

The main components of the tourism ontology are concepts, relations, instances, and axioms. A concept represents a set or class of entities within the tourism domain.

Each class defined by an ontology describes common characteristics of individuals. OWL classes permit much greater expressiveness than RDF Schema classes. Consequently, OWL has created their own classes, owl:Class. owl:Thing is a predefined OWL class. All instances are members of owl:Thing. The owl:Nothing is also a predefined class and represents the empty class. Each defined class is of type owl:Class. What, Where, and When are examples of classes used in our e-tourism ontology. These concepts are represented in OWL in the following way:

```
<owl:Class rdf:ID="What"/>
```

```
<owl:Class rdf:ID="Where"/>
<owl:Class rdf:ID="When"/>
<owl:Class rdf:ID="Tourist">
  <rdfs:comment> Describes a tourist </rdfs:comment>
</owl:Class>
```

The class What refers to activities that tourists can carry out, such as golf, sightseeing, shopping, or visiting a theatre. The class Where refers to the places where a tourist can stay (such as a Hotel) and places where he can carry out an activity. Examples of infrastructures that provide the means for exerting an activity include restaurants, cinemas, or museums. The class When refers to the time when a tourist can carry out an activity at a certain place.

The ontology also includes relations which describe the interactions between classes or properties. A class hierarchy may be defined by stating that a class is a subclass (owl:subClassOf) of another class. For example, in the tourism domain, the class Squash, Paintball, and Golf are subclasses of the class What. These three classes and their relationship are defined using the OWL vocabulary:

```
...
<owl:Class rdf:ID="Squash">
  <rdfs:comment> Squash is an activity a tourist
                 can carry out
  </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#What"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Paintball">
  <rdfs:subClassOf rdf:resource="#What"/>
</owl:Class>

<owl:Class rdf:ID="Golf">
  <rdfs:subClassOf rdf:resource="#What"/>
...
</owl:Class>
```

The first statement states that in order to be an instance of the class Squash, an individual must also be an instance of the class What. However,

there may be instances of the class What that are not instances of Squash. Thus being a What is a necessary condition for Squash, but is not sufficient.

In our example, we have defined the three subclasses using two different notations. The semantics of the two notations are the same. Nevertheless, we prefer the second one, since it is easier to read.

Two classes can be made equivalent using the assertion owl:equivalentClass. This property, when applied to two classes, A and B, is to be interpreted as "classes A and B contain exactly the same set of individuals." This property is especially useful to be able to indicate that a particular class in an ontology is equivalent to a class defined in a second ontology. For example, the class What can be defined equivalent to the class Activity:

```
<owl:Class rdf:ID="Activity"/>
<owl:Class rdf:ID="What">
  <rdfs:comment> Describes an activity a tourist
                 can carry out
  </rdfs:comment>
  <owl:equivalentClass rdf:resource="#Activity"/>
</owl:Class>
```

It is also possible to state that two classes are disjoint using the owl:disjointWith statement. This statement guarantees that an individual that is a member of one class cannot simultaneously be an instance of another class. For example, we can express that the activity Golf is disjoint with the activities Squash and Paintball.

```
<owl:Class rdf:ID="Golf">
  <rdfs:comment> Golf is an activity a tourist
                 can carry out
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#What"/>
  <owl:disjointWith rdf:resource="#Squash"/>
  <owl:disjointWith rdf:resource="#Paintball"/>
</owl:Class>
```

This example expresses that instances belonging to one subclass, e.g. Golf, cannot belong to another subclass, e.g. Squash or Paintball. A reasoning engine could identify an inconsistency when an individual of the class Golf is stated to be an instance of the class Squash. The reasoning engine could also deduce that if G is an instance of Golf, then G is not an instance of Squash or Paintball.

5.3 Complex Classes

The OWL language provides a set of statements for building complex class descriptions from simpler ones by allowing the specification of the Boolean combination of classes. Boolean connectives (owl:complementOf, owl:intersectionOf, and owl:unionOf) combine class descriptions using logical connectives. For example, two classes, A and B, can be intersected yielding a new class C. Additional set operators include the union and the complement. With OWL Lite only the intersection of classes is allowed.

The owl:complementOf element is applied to a single class and describes the set of all individuals which are known not to be instances of the class. For example, we can state that tourists from the European Union are not tourists from the non-European Union countries.

```
<owl:Class rdf:ID="EUTourist">
  <rdfs:subClassOf rdf:resource="#Tourist"/>
</owl:Class>

<owl:Class rdf:ID="NonEUTourist">
  <rdfs:subClassOf rdf:resource="#Tourist"/>
  <owl:complementOf rdf:resource="#EUTourist" />
</owl:Class>
```

In this example, the class NonEUTourist refers to a very large set of individuals. The class has as its members all individuals that do not belong to the EUTourist class. This means that an individual of any class, such as Locals, Countries, and SiteSeeingPackage, other than the class EUTourist, belongs to the class NonEUTourist.

As the name suggests, the owl:intersectionOf, can be used to intersect two classes, A and B. The new class includes the individuals that were both in class A and in class B.

This element is often used with the owl:Restriction element. For example, taking the intersection of the class of tourist with the anonymous class of people that are senior citizens describes the class of senior tourists.

```
<owl:Class rdf:ID="seniorTourists">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Tourist"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#category"/>
      <owl:hasValue rdf:resource="#Senior"/>
    </owl:Restriction>
```

```
    </owl:intersectionOf>
  </owl:Class>
```

The individuals who are members of the seniorTourists class are precisely those individuals who are members of both the class #Tourist and the anonymous class created by the restriction on the property #category. While not shown in this example, the category of a tourist is divided into Junior, Young, and Senior. Restrictions will be discussed later.

The element owl:unionOf when applied to two classes, A and B, works in a similar way to the owl:intersectionOf element, but creates a new class which has as its members all individuals that are in class A or in class B. The new class is equal to the union of the two initial classes. For example, the individuals of the class OutdoorSport are the union of all the individuals that belong to the class Golf or to the class Paintball.

```
<owl:Class rdf:ID="OutdoorSport">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Golf"/>
    <owl:Class rdf:about="#Paintball"/>
  </owl:unionOf>
</owl:Class>
```

In other words, the individuals who are members of the class OutdoorSport are those individuals who are either members of the class Golf or the class Paintball.

5.4 Enumeration

An OWL class can be described by enumeration of the individuals that belong to the class. The members of the class are exactly the set of enumerated individuals. This is achieve using the element owl:oneOf and enables a class to be described by exhaustively enumerating its individuals. This element is not allowed with OWL Lite. For example, the class of HotelRoomView can be described by enumerating it individuals: Sea, Mountain, and City.

```
<owl:Class rdf:ID="HotelRoomView"/>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:ID="#Sea"/>
    <owl:Thing rdf:ID="#Mountain"/>
    <owl:Thing rdf:ID="#City"/>
  </owl:oneOf>
```

```
</owl:Class>
```

5.5 Properties

5.5.1 Simple Properties

OWL can define the properties of classes. The OWL property is not very different from a RDFS property. They both use the rdfs:domain and rdfs:range elements. Simple properties can be defined using: owl:ObjectProperty and owl:DatatypeProperty.

Object properties link individuals to individuals. They relate an instance of a class to an instance of another class. The other class can actually be the same class.

For example, the object property hasActivity related the class Where with the class What. This means that a place (i.e., an individual of the class Where) may supply a kind of activity (i.e., an individual of the class What) to its customer, such as Golf and Paintball. The first related class is called the domain, while the second is called the range:

```
<owl:ObjectProperty rdf:ID="hasActivity">
  <rdfs:domain rdf:resource="#Where"/>
  <rdfs:range rdf:resource="#What"/>
</owl:ObjectProperty>
```

Datatype properties link individuals to data values and can be used to restrict an individual member of a class to RDF literals and XML Schema datatypes. Since OWL does not include any data types, it allows the XML Schema data types to be used. All OWL reasoners are required to support the xsd:integer and xsd:string datatypes. In the following example, the year a tourist was born is specified using the &xsd;positiveInteger data type from the XML Schema.

```
<owl:DatatypeProperty rdf:ID="ageYear">
  <rdfs:comment> The year a tourist was born
  </rdfs:comment>
  <rdfs:range rdf:resource= "&xsd;positiveInteger"/>
  <rdfs:domain rdf:resource="#Tourist"/>
</owl:DatatypeProperty>
```

5.5.2 Property Characteristics

Property characteristics allow data to be made more expressive in such a way that reasoning engines can carry out powerful inference. They enhance reasoning by extending the meaning behind relationships. In OWL, it is possible to define relations from one property to other properties. Two examples are the elements owl:equivalentProperty and owl:inverseOf.

The equivalence of properties is defined using the owl:equivalentProperty element. Property equivalence is not the same as property equality. Equivalent properties have the same property extension, but may have different meanings. The following example expresses that stating that "a Person plays a sport" is equivalent to stating that "a Person engages in a sport".

```
<owl:ObjectProperty rdf:ID="plays">
  <rdfs:domain rdf:resource="#Person"/>
...
  <owl:equivalentProperty rdf:resource="#engages"/>
</owl:ObjectProperty>
```

The owl:inverseOf construct can be used to define inverse relation between properties. If the property P' is stated to be the inverse of the property P'', then if X'' is related to Y'' by the P'' property, then Y'' is related to X'' by the P' property. For example, "a tourist plays an activity" and "an activity isPlayedBy a tourist" are cases of an inverse relation between properties. In such a scenario, if the tourist John plays the activity Golf, then a reasoner may infer that Golf isPlayedBy John. This can be expressed formally in OWL as:

```
<owl:ObjectProperty rdf:ID="isPlayedBy">
  <owl:inverseOf rdf:resource="#plays"/>
</owl:ObjectProperty>
```

Functional properties (owl:FunctionalProperty) express the fact that a property may have no more than one value for each instance. Functional properties have a unique value or no values, i.e. the property's minimum cardinality is zero and its maximum cardinality is 1. If an individual instance of Tourist has the PassportID property, then that individual may not have more than one ID. However, this does not state that every Tourist must have at least one passport ID. This is illustrated in the following example with the hasPassportID property, which ensures that a Tourist has only one passport ID:

```
<owl:ObjectProperty rdf:ID="hasPassportID">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Tourist"/>
  <rdfs:range rdf:resource="#PassportID"/>
</owl:ObjectProperty>
```

The same semantic can be expressed as:

```
<owl:ObjectProperty rdf:ID="hasPassportID">
  <rdfs:domain rdf:resource="#Tourist"/>
  <rdfs:range rdf:resource="#PassportID"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasPassportID"/>
```

Common examples of functional properties include age, height, date of birth, sex, marital status, etc.

Properties may be stated to be inverse functional with the element owl:InverseFunctionalProperty. If a property is inverse functional then the inverse of the property is functional and the inverse functional property defines a property for which two different objects cannot have the same value. The inverse of the property has at most one value. The following example states that the property isThePassportIDof is to be inverse functional:

```
<owl:InverseFunctionalProperty
                              rdf:ID="isThePassportIDof">
  <rdfs:domain rdf:resource="#PassportID"/>
  <rdfs:range rdf:resource="#Tourist"/>
</owl:InverseFunctionalProperty>
```

Therefore, there can only be one passport ID for a tourist. The inverse property of isThePassportIDof, i.e. the functional property hasPassportID has at most one value.

A reasoning engine can infer that no two tourists can have the same passport ID and that if two tourists have the same passport number, then they refer to the same individual.

FunctionalProperty and InverseFunctionalProperty can be used to relate resources to resources, or resources to an RDF Schema Literal or an XML Schema datatype.

Properties may be also stated to be symmetric. The symmetric property (owl:SymmetricProperty) is interpreted as follows: if the pair (x, y) is an instance of A, then the pair (y, x) is also an instance of A.

For example, the property b2bLink of the class Hotel of our e-tourism ontology may be stated to be a symmetric property:

```
<owl:ObjectProperty rdf:ID="b2bLink">
  <rdf:type rdf:resource="&owl;SymmetricProperty"/>
  <rdfs:domain rdf:resource="#Hotel"/>
  <rdfs:range rdf:resource="#LeisureOrganization"/>
</owl:ObjectProperty>
```

This expresses the fact that a Hotel can establish B2B (Business-to-Business) links with several leisure organizations from the tourism industry. For example, a Hotel can establish a B2B link with a Golf course and a SPA. When a reasoner is given the fact that a Hotel A has established a B2B link with a Golf course B, the reasoner can infer that the Golf course B has also a B2B link with the Hotel A.

When a property is stated to be transitive with the element owl:TransitiveProperty, then if the pair (x, y) is an instance of the transitive property P, and the pair (y, z) is an instance of P, we can infer the pair (x, z) is also an instance of P

For example, if busTour is stated to be transitive, and if there is a bus tour from Funchal to Porto Moniz and there is a bus tour from Porto Moniz to São Vicente, then a reasoner can infer that there is a bus tour from Funchal to São Vicente. Funchal, Porto Moniz, and São Vicente are individuals of the class Where. This is expressed in OWL in the following way:

```
<owl:TransitiveProperty rdf:ID="busTour">
  <rdfs:domain rdf:resource="#Where"/>
  <rdfs:range rdf:resource="#Where"/>
</owl:TransitiveProperty>
```

Or equivalently:

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

Both the owl:SymmetricProperty and owl:TransitiveProperty properties are used to relate resources to resources.

5.6 Property Restrictions

Restrictions differ from characteristics since restrictions apply to properties with specific values. Property restrictions allow specifying a class for which its instances satisfy a condition. A restriction is achieved through the owl:Restriction element which contains an owl:onProperty element and one or more restriction declarations. Examples of restrictions include owl:allValuesFrom (specifies universal quantification), owl:hasValue (specifies a specific value), and owl:someValuesFrom (specifies existential quantification).

The owl:allValuesFrom element is stated on a property with respect to a class. A class may have a property P restricted to have all the values from the class C, i.e. the constraint demands that all values of P should be of type C (if no such values exist, the constraint is trivially true). Let us see an example to better understand this concept:

```
<owl:Class rdf:ID="TouristOutdoorSportPlayer">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#plays"/>
      <owl:allValuesFrom
                  rdf:resource="#OutdoorSport"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The individuals that are members of the class TouristOutdoorSportPlayer are those such that if there is an object that is related to them via the #plays property, then it must be #OutdoorSport. No assertion about the existence of the relationship #plays is made, but if the relationship holds then the related object must be of the class #OutdoorSport.

Using the owl:hasValue element, a property can be required to have a specific value. For example, individuals of the class FunchalSiteSeeing can be characterized as those places that have 9000 as a value of their zip code. This is expressed with the following statements:

```
<owl:Class rdf:ID="FunchalSiteSeeing">
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasZipCode"/>
        <owl:hasValue rdf:datatype="&xsd;string">
          9000
        </owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

In terms of logic, the owl:someValuesFrom element allows expression of existential quantification. This element describes those individuals that have a relationship with other individuals of a particular class. Unlike owl:allValuesFrom, owl:someValuesFrom does not restrict all the values of the property to be individuals of the same class. When owl:someValuesFrom is stated on a property P with respect to a class C, it specifies that at least one value for that property is of a certain type.

For example, the class TouristGolfPlayer may have a owl:someValuesFrom restriction on the #plays property that states that some value for the plays property should be an instance of the class Golf. This expresses the fact that any tourist can play multiple sports (e.g. Golf, PaintBall, Tennis, etc.) as long as one or more is an instance of the class Golf.

```
<owl:Class rdf:ID="TouristGolfPlayer">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Tourist"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#plays"/>
      <owl:someValuesFrom rdf:resource="#Golf"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

The individuals that are members of the class TouristGolfPlayer are those that are related via the #plays property to at least one instance of the Golf class. The owl:someValuesFrom element makes no restriction about other relationships that may be present. Therefore, an individual of the class TouristGolfPlayer may play other sports.

5.7 Cardinality Restrictions

Cardinality restrictions are also property restrictions. In OWL, three different cardinality restrictions exist:

- owl:maxCardinality – specifies the maximum number of individuals,
- owl:minCardinality – specifies the minimum number of individuals, and
- owl:cardinality – specifies the exact number of individuals.

The element owl:maxCardinality: is stated on a property P with respect to a particular class C. If a owl:maxCardinality with the value *n* is stated on a property with respect to a class, then any instance of that class will be related to at most n individuals by property P. The variable *n* should be a non-negative integer.

For example, the property #visitLocal of the class SiteSeeingPackage may have a maximum cardinality of 10 since it is considered that a site seeing package should not include more than 10 places to visit.

```
<owl:Class rdf:ID="SiteSeeingPackage">
...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#visitLocal"/>
      <owl:maxCardinality rdf:datatype=
                          "&xsd;nonNegativeInteger"> 10
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The element owl:minCardinality is very similar to the element owl:maxCardinality. As the name suggests, the only difference lies in the fact that it specified a lower boundary for the cardinality of a property P of a class C. The following example shows that the property visitLocal of the class SiteSeeingPackage has a minimum cardinality of 2. It expressed that a site seeing package should include the visit to at least 2 site seeing locals.

```
<owl:Class rdf:ID="SiteSeeingPackage">
...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#visitLocal"/>
    <owl:minCardinality
```

```
            rdf:datatype="&xsd;nonNegativeInteger"> 2
      </owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

The owl:cardinality, the last cardinality restriction statement, is a useful element when it is necessary to expresse that a property has a minimum cardinality which is equal to the maximum cardinality. This is a convenience element.

It should be noticed that when using OWL Lite the cardinality elements, owl:maxCardinality, owl:minCardinality, and owl:cardinality, can only specify the values 0 and 1. On the other hand, OWL Full allows cardinality statements for arbitrary non-negative integers. Furthermore, when using OWL DL, no cardinality restrictions may be placed on transitive properties

6. PUTTING ALL TOGETHER: THE E-TOURISM ONTOLOGY

The following example describes the e-tourism ontology. This ontology can be use to integrate tourist information systems or simply serve as a schema to carry out inferencing.

```
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY owl "http://www.w3.org/2002/07/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#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns ="http://dme.uma.pt/jcardoso/owl/e-tourism#"
  xml:base="http://dme.uma.pt/jcardoso/owl/e-
tourism#">

<owl:Ontology rdf:about="">
  <rdfs:comment>E-Tourism OWL Ontology
  </rdfs:comment>
```

```
    <owl:versionInfo> v.1 2005-10-25
    </owl:versionInfo>
    <owl:priorVersion>
    <owl:Ontology rdf:about=
    "http://dme.uma.pt/jcardoso/owl/tourism.owl"/>
    </owl:priorVersion>
    <owl:backwardCompatibleWith rdf:resource=
          "http://dme.uma.pt/jcardoso/owl/tourism.owl"/>
    <rdfs:label>E-Tourism Ontology</rdfs:label>
  </owl:Ontology>

  <owl:Class rdf:ID="When">
    <rdfs:comment> Describes when a tourist can carry
                   out a particular activity
    </rdfs:comment>
  </owl:Class>

  <owl:Class rdf:ID="Place"/>
  <owl:Class rdf:ID="Where">
    <rdfs:comment> Describes where a tourist can carry
                    out a particular activity or stay
                    overnight
    </rdfs:comment>
    <owl:equivalentClass rdf:resource="#Place"/>
  </owl:Class>

  <owl:Class rdf:ID="Activity"/>
  <owl:Class rdf:ID="What">
    <rdfs:comment> Describes an activity a tourist
                   can carry out
    </rdfs:comment>
    <owl:equivalentClass rdf:resource="#Activity"/>
  </owl:Class>

  <owl:Class rdf:ID="Tourist">
    <rdfs:comment> Describes a tourist. Every tourist
                   is a person
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>

  <owl:Class rdf:ID="EUTourist">
```

```
    <rdfs:subClassOf rdf:resource="#Tourist"/>
  </owl:Class>

  <owl:Class rdf:ID="NonEUTourist">
    <rdfs:subClassOf rdf:resource="#Tourist"/>
    <owl:complementOf rdf:resource="#EUTourist" />
  </owl:Class>

  <owl:Class rdf:ID="PassportID">
    <rdfs:comment> Tourists have passports with an ID
    </rdfs:comment>
  </owl:Class>

  <owl:Class rdf:ID="Hotel">
    <rdfs:comment> Hotel is a place where a tourist
                   can stay overnight
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Where"/>
  </owl:Class>

  <owl:Class rdf:ID="HotelRoomView">
    <rdfs:comment> Enumerates the views a hotel room
                   can have
    </rdfs:comment>
    <owl:oneOf rdf:parseType="Collection">
      <owl:Thing rdf:about="#Sea"/>
      <owl:Thing rdf:about="#Mountain"/>
      <owl:Thing rdf:about="#City"/>
    </owl:oneOf>
  </owl:Class>

  <owl:Class rdf:ID="LeisureOrganization">
    <rdfs:comment> A leisure organization provides
                  activities that tourists can carry out
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Where"/>
  </owl:Class>

  <owl:Class rdf:ID="Squash">
    <rdfs:comment> Squash is an activity a tourist
                   can carry out
    </rdfs:comment>
```

```
  <rdfs:subClassOf rdf:resource="#What"/>
</owl:Class>

<owl:Class rdf:ID="Paintball">
  <rdfs:comment> Paintball is also an activity
                 a tourist can carry out
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#What"/>
</owl:Class>

<owl:Class rdf:ID="Golf">
  <rdfs:comment> Golf is an activity a tourist
                 can carry out
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#What"/>
  <owl:disjointWith rdf:resource="#Squash"/>
  <owl:disjointWith rdf:resource="#Paintball"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="ageYear">
  <rdfs:comment> The year a tourist was born
  </rdfs:comment>
  <rdfs:range rdf:resource= "&xsd;positiveInteger"/>
  <rdfs:domain rdf:resource="#Tourist"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="category">
  <rdfs:comment> The category of a tourist (e.g.
                 Junior, Young, Senior)
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Tourist"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasActivity">
  <rdfs:comment> Describes an activity that can be
                 carried out a certain place
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Where"/>
  <rdfs:range rdf:resource="#What"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="hasZipCode">
```

```
    <rdfs:comment> Each place has a zip code
    </rdfs:comment>
    <rdfs:domain rdf:resource="#Where"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:ObjectProperty rdf:ID="plays">
    <rdfs:comment> The activity that a person
                   carries out
    </rdfs:comment>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Squash"/>
          <owl:Class rdf:about="#Golf"/>
          <owl:Class rdf:about="#Paintball"/>
          </owl:unionOf>
      </owl:Class>
    </rdfs:range>
    <owl:equivalentProperty rdf:resource="#engages"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="isPlayedBy">
    <owl:inverseOf rdf:resource="#plays"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="hasPassportID">
    <rdfs:comment> Carrying out an activity or engaging
                   in an activity are two equivalent
                   properties
    </rdfs:comment>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Tourist"/>
    <rdfs:range rdf:resource="#PassportID"/>
  </owl:ObjectProperty>

  <owl:InverseFunctionalProperty
                            rdf:ID="isthePassportIDof">
    <rdfs:domain rdf:resource="#PassportID"/>
    <rdfs:range rdf:resource="#Tourist"/>
  </owl:InverseFunctionalProperty>
```

```
<owl:ObjectProperty rdf:ID="b2bLink">
  <rdfs:comment> Hotels establish B2B links with
                  leisure organizations
  </rdfs:comment>
  <rdf:type rdf:resource="&owl;SymmetricProperty"/>
  <rdfs:domain rdf:resource="#Hotel"/>
  <rdfs:range rdf:resource="#LeisureOrganization"/>
</owl:ObjectProperty>

<owl:TransitiveProperty rdf:ID="busTour">
  <rdfs:comment> Bus tours are offered from place A
                 to place B
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Where"/>
  <rdfs:range rdf:resource="#Where"/>
</owl:TransitiveProperty>

<owl:Class rdf:ID="GoodWeather"/>
<owl:Class rdf:ID="BadWeather"/>
<owl:Class rdf:ID="AverageWeather"/>

<owl:ObjectProperty rdf:ID="hasWeather">
  <rdfs:comment> Describes the weather at a
                 particular place
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Where"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="PlacesWithGoodWeather">
  <rdfs:comment> Describes the tourist places with a
                 good weather
  </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasWeather"/>
      <owl:allValuesFrom rdf:resource="#GoodWeather"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="FunchalSiteSeeing">
```

```
    <rdfs:comment> Describes the places that tourist can
  see in Funchal. These places have the zip code 9000,
  i.e. the city of Funchal.
    </rdfs:comment>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasZipCode"/>
        <owl:hasValue rdf:datatype="&xsd;string"> 9000
        </owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="SiteSeeingPackage">
    <rdfs:comment> A site seeing package should include
  at least 2 places to visit, but no more than 10.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Where"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasZipCode"/>
      <owl:minCardinality
              rdf:datatype="&xsd;nonNegativeInteger"> 2
      </owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasZipCode"/>
        <owl:maxCardinality
              rdf:datatype="&xsd;nonNegativeInteger"> 10
        </owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="TouristGolfPlayer">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Tourist"/>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#engages"/>
        <owl:someValuesFrom rdf:resource="#Golf"/>
```

```
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>

  <owl:Class rdf:ID="TouristOutdoorSportPlayer">
    <rdfs:comment> Describes the tourist places with a
                   good weather
    </rdfs:comment>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#engages"/>
        <owl:allValuesFrom
                          rdf:resource="#OutdoorSport"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="OutdoorSport">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Golf"/>
      <owl:Class rdf:about="#Paintball"/>
    </owl:unionOf>
  </owl:Class>

  </rdf:RDF>
```

7. QUESTIONS FOR DISCUSSION

Beginner:

1. RDF, RDFS, and OWL are languages that correspond to layers of the semantic Web stack and are built on top of XML. Why is XML not itself a semantic language?
2. What are the limitations of RDFS that make it not sufficiently expressive to describe the semantics of Web resources?

Intermediate:

1. Two instance with a different rdf:ID can actually represent the same individual. With OWL, how can you make it explicit that the two instances are different?

2. Use the XMLSchema to define a complex data type to model a student record (e.g. name, degree, ID, etc.) and reference this data type within an OWL ontology.

Advanced:
1. OWL is based on the open world assumption. Identify the characteristics that do not make OWL follow the closed world assumption.
2. Describe a scenario that illustrates how reasoning engines can use the owl:unionOf and owl:intersectionOf elements to carry out inference.

Practical Exercises:
1. Select a Web site, such as www.amazon.com, and develop an OWL ontology to model the information present on its main page.
2. Validate the OWL ontology developed with an OWL validator (e.g. http://owl.bbn.com/validator/)
3. Use a reasoning engine, such as JESS (herzberg.ca.sandia.gov/jess/) to infer knowledge from the developed ontology.

8. SUGGESTED ADDITIONAL READING

- Antoniou, G. and van Harmelen, F. *A semantic Web primer.* Cambridge, MA: MIT Press, 2004. pp. 238: This book is a good introduction to Semantic Web languages.
- Shelley Powers, *Practical RDF*, O'Reilly, 2003, pp. 331: This book covers RDF, RDFS, and OWL. It provides a good source of information for those interested in programming with RDF with Perl, PHP, Java, and Python.
- Seffen Staab, *Ontology Handbook*, Springer, 2003, pp. 499: This book covers provides a good introduction to Description Logics and OWL.
- OWL Overview – http://www.w3.org/TR/owl-features/
- OWL Reference – http://www.w3.org/TR/owl-ref/
- OWL Guide – http://www.w3.org/TR/owl-guide/

9. REFERENCES

Berners-Lee, T., J. Hendler, et al. (2001). The Semantic Web. Scientific American. **May 2001**.

Berners-Lee, T., J. Hendler, et al. (2001). The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American.

DAML (2001). DAML+OIL, http://www.daml.org/language/.

Decker, S., S. Melnik, et al. (2000). "The Semantic Web: The Roles of XML and RDF." Internet Computing **4**(5): 63-74.

Horrocks, I., F. v. Harmelen, et al. (2001). DAML+OIL, DAML.

Jasper, R. and M. Uschold (1999). A framework for understanding and classifying ontology applications. IJCAI99 Workshop on Ontologies and Problem-Solving Methods.Vol: pp.

OWL (2004). OWL Web Ontology Language Reference, W3C Recommendation, World Wide Web Consortium, http://www.w3.org/TR/owl-ref/. **2004**.

RDF (2002). Resource Description Framework (RDF), http://www.w3.org/RDF/.

RDFS (2004). RDF Vocabulary Description Language 1.0: RDF Schema, W3C, http://www.w3.org/TR/rdf-schema/.