Introduction to Semantic Web Ontology Languages

Grigoris Antoniou[1], Enrico Franconi[2], and Frank van Harmelen[3]

[1] ICS-FORTH, Greece
antoniou@icsforth.gr
[2] Faculty of Computer Science, Free University of Bozen–Bolzano, Italy
franconi@inf.unibz.it
[3] Department of Computer Science, Vrije Universiteti Amsterdam, Netherlands
frankh@cs.vu.nl

Abstract. The aim of this chapter is to give a general introduction to some of the ontology languages that play a prominent role on the Semantic Web, and to discuss the formal foundations of these languages. Web ontology languages will be the main carriers of the information that we will want to share and integrate.

1 Organisation of This Chapter

In section 2 we discuss general issues and requirements for Web ontology languages, including the semantics issues. We then describe briefly the most important ontology languages in the design of the Semantic Web, namely RDF Schema in section 3 and OWL in section 4. Section 5 contains a brief comparison with other ontology languages. A brief introduction to description logics and their relation to the OWL family of web ontology languages is included. The chapter is concluded by a discussion on the importance of having correct and complete inference engines for web ontology languages.

2 On Web Ontology Languages

Even though ontologies have a long history in Artificial Intelligence (AI), the meaning of this concept still generates a lot of controversy in discussions, both within and outside of AI. We follow the classical AI definition: an ontology is a *formal specification of a conceptualisation*, that is, an abstract and simplified view of the world that we wish to represent, described in a language that is equipped with a formal semantics. In knowledge representation, an ontology is a description of the concepts and relationships in an application domain. Depending on the users of this ontology, such a description must be understandable by humans and/or by software agents. In many other field – such as in information systems and databases, and in software engineering – an ontology would be called a *conceptual schema*. An ontology is formal, since its understanding should be non ambiguous, both from the syntactic and the semantic point of views.

Researchers in AI were the first to develop ontologies with the purpose of facilitating automated knowledge sharing. Since the beginning of the 90's, ontologies have become a popular research topic, and several AI research communities, including knowledge engineering, knowledge acquisition, natural language processing, and knowledge representation, have investigated them. More recently, the notion of an ontology is becoming widespread in fields such as intelligent information integration, cooperative information systems, information retrieval, digital libraries, e-commerce, and knowledge management. Ontologies are widely regarded as one of the foundational technologies for the Semantic Web: when annotating web documents with machine-interpretable information concerning their *content*, the meaning of the terms used in such an annotation should be fixed in a (shared) ontology. Research in the Semantic Web has led to the standardisation of specific web ontology languages.

An ontology language is a mean to specify at an abstract level – that is, at a *conceptual* level – what is necessarily true in the domain of interest. More precisely, we can say that an ontology language should be able to express *constraints*, which declare what should necessarily hold in any possible concrete instantiation of the domain. In the following, we will introduce various ways to impose constraints over domains, by means of statements expressed is some suitable ontology language.

2.1 What Are Ontology Languages

How do we describe a particular domain? Let us consider the domain of courses and lecturers at Griffith University. First we have to specify the "things" we want to talk about. Here we will make a first, fundamental distinction. On one hand we want to talk about particular lecturers, such as David Billington, and particular courses, such as Discrete Mathematics. But we also want to talk about courses, first year courses, lecturers, professors etc. What is the difference? In the first case we talk about *individual objects* (resources), in the second we talk about *classes* (also called *concepts*) which define types of objects.

A class can be thought of as a set of elements, called the *extension* of the class. Individual objects that belong to a class are referred to as *instances* of that class.

An important use of classes is to *impose restrictions* on what can be stated. In programming languages, *typing* is used to prevent nonsense from being written (such as $A + 1$, where A is an array; we lay down that the arguments of $+$ must be numbers). The same is needed in RDF. After all, we would like to disallow statements such as:

– Discrete Mathematics is taught by Concrete Mathematics.
– Room MZH5760 is taught by David Billington.

The first statement is non-sensical because we want courses to be taught by lecturers only. This imposes a restriction on the values of the property "is taught by". In mathematical terms, we restrict the *range* of the property.

The second statement is non-sensical because only courses can be taught. This imposes a restriction on the objects to which the property can be applied. In mathematical terms, we restrict the *domain* of the property.

Class Hierarchies. Once we have classes we would also like to establish relationships between them. For example, suppose that we have classes for

– staff members
– academic staff members
– professors
– associate professors
– assistant professors
– administrative staff members
– technical support staff members.

These classes are not unrelated to each other. For example, every professor is an academic staff member. We say that professor is a *subclass* of academic staff member, or equivalently, that academic staff member is a *superclass* of professor. The subclass relationship is also called *subsumption*.

The subclass relationship defines a hierarchy of classes. In general, A is a subclass of B if every instance of A is also an instance of B.

A hierarchical organisation of classes has a very important practical significance, which we outline now. Consider the range restriction

Courses must be taught by academic staff members only.

Suppose Michael Maher was defined as a professor. Then, according to the restriction above, he is not allowed to teach courses. The reason is that there is no statement which specifies that Michael Maher is also an academic staff member. Obviously it would be highly counterintuitive to overcome this difficulty by adding that statement to our description. Instead we would like Michael Maher to *inherit* the ability to teach from the class of academic staff members.

Property Hierarchies. We saw that hierarchical relationships between classes can be defined. The same can be done for properties. For example, "is taught by" is a *subproperty* of "involves". If a course c is taught by an academic staff member a, then c also involves a. The converse is not necessarily true. For example, a may be the convenor of the course, or a tutor who marks student homework, but does not teach c.

In general, P is a subproperty of Q if two objects are related by Q whenever they are related by P.

Summary. As a consequence of the discussion above, (Web) ontology languages consist of:

- the important concepts (classes) of a domain
- important relationships between these concepts. These can be hierarchical (subclass relationships), other predefined relationships contained in the ontology language, or user defined (properties).
- further constraints on what can be expressed (e.g. domain and range restrictions, cardinality constraints etc.).

2.2 Formal Semantics

Ontology languages allow users to write explicit, formal conceptualisations of domains models. The main requirements are:

1. a well-defined syntax
2. a well-defined semantics
3. efficient reasoning support
4. sufficient expressive power
5. convenience of expression.

The importance of a *well-defined syntax* is clear, and known from the area of programming languages; it is a necessary condition for *machine-processing* of information. Web ontology languages have a syntax based on XML, though they may also have other kinds of syntaxes.

Of course it is questionable whether the XML-based syntax is very user-friendly, there are alternatives better suitable for humans. However this drawback is not very significant, because ultimately users will be developing their ontologies using authoring tools, or more generally *ontology development tools*, instead of writing them directly in the Web ontology language.

Formal semantics describes precisely the meaning of knowledge. “Precisely” here means that the semantics does not refer to subjective intuitions, nor is it open to different interpretations by different persons (or machines). The importance of formal semantics is well-established in the domain of mathematical logic. In the context of ontology languages, the semantics enforces the meaning of the expressed knowledge as a set of constraints over the domain. Any possible instantiation of the domain should necessarily conform to the constraints expressed by the ontology.

Given a statement in an ontology, the role of the semantics is to devise precisely which are the *models* of the statement, i.e., all the possible instantiations of the domain that are compatible with the statement. We say that a statement is *true* in an instantiation of the domain if this instantiation is compatible with the statement; the instantiation of the domain in which a statement is true is of course a model of the statement, and viceversa. So, an ontology will itself devise a set of models, which is the intersection among all the models of each statement in the ontology. The models of an ontology represent the only possible realisable situations.

For example, if an ontology states that professor is a subclass of academic staff member (i.e., in any possible situation, each professor is also an academic staff member), and if it is known that Michael Maher is a professor (i.e., Michael

Maher is an instance of the professor class), then in any possible situation it is necessarily true that Michael Maher is an academic staff member, since the situation in which he would not be an academic staff member is incompatible with the constraints expressed in the ontology.

If we understand that an ontology language talks basically about classes, properties and objects of a domain, then a model (i.e., a specific instantiation of the domain) is nothing else than the precise characterisation for each objects of the classes it is instance of, and of the properties it participates to. So, in the above example, in any model of the ontology Michael Maher should be an instance of the academic staff member class.

2.3 Reasoning

The fact that the formal semantics associates to an ontology a set of models, allows us to define the notion of *deduction*. Given an ontology, we say that an additional statement can be deduced from the ontology if it is true in all the models of the ontology. This definition of deduction comes from logic and it is very general but also very strict: if a statement is not true in all the models of an ontology, then it is not a valid deduction from it. The process of deriving valid deductions from an ontology is called *reasoning*.

If we consider the typical statements of web ontology languages, the following deductions ("inferences") can be introduced:

– *Class membership.* We want to deduce whether an object is instance of a class. For example, if in the ontology it is stated that `Michael Maher` is an instance of a class `Professor`, and that `Professor` is a subclass of the `Academic Staff Member` class, then we can infer that `Michael Maher` is an instance of `Academic Staff Member`, because this latter statement is true in all the models of the ontology, as we have explained above.
– *Classification:* We want to deduce all the subclass relationships between the existing classes in the ontology. For example, if in the ontology it is stated that the class `Teaching Assistant` is a subclass of the `Professor` class, and that `Professor` is a subclass of the `Academic Staff Member` class, then we can infer that `Teaching Assistant` is a subclass of `Academic Staff Member`. This deduction holds since in any model of the ontology the extension of `Teaching Assistant` is a subset of the extension of `Professor`, and the extension of `Professor` is a subset of the extension of `Academic Staff Member`. Therefore, in any model the extension of `Teaching Assistant` is a subset of the extension of `Academic Staff Member`, and in any model the statement that `Teaching Assistant` is a subclass of `Academic Staff Member` is true.
– *Equivalence of classes.* We want to deduce whether two classes are equivalent, i.e., they have the same extension. For example, if class `Professor` is equivalent to class `Lecturer`, and class `Lecturer` is equivalent to class `Teacher`, then `Professor` is equivalent to `Teacher`, too.
– *Consistency of a class.* We want to check that some class does not have necessarily an empty extension. For example, given an ontology in which the class `Working-Student` is defined to be a subclass of two disjoint classes

`Student` and `Professor`, it can be inferred that the class `Working-Student` is inconsistent, since in every model of the ontology its extension is empty. In fact, any instance of `Working-Student` would violate the constraints imposed by the ontology (namely, that there is no common instance between the two classes). In this case, it would be possible to remove the inconsistency for the `Working-Student` class by removing from the ontology the disjointness statement between `Student` and `Professor`.

– *Consistency of the ontology.* We want to check that the ontology admits at least a model, i.e., there is at least a possibility to have an instantiation of the domain compatible with the ontology. For example, suppose we have declared in the ontology
  1. that `John` is an instance of both the class `Student` and the class `Professor`, and
  2. that `Student` and `Professor` are two disjoint classes.

  Then we have an inconsistency because the two constraints can not be satisfied simultaneously. Statement 2 says that the extensions of the two classes can not have any element in common, since they are disjoint, but statement 1 says that `John` is an instance of both classes. This clearly indicates that there is an error in the ontology, since it does not represent any possible situation.

In designing an ontology language one should be aware of the *tradeoff between expressive power and efficiency of reasoning.* Generally speaking, the richer the language is, the more inefficient the reasoning support becomes, often crossing the border of non-computability. Thus we need a compromise, a language that can be supported by reasonably efficient reasoners, while being sufficiently expressive to express large classes of ontologies and knowledge.

Various methodologies are being developed on how to build a "good" ontology. These approaches may differ in many aspects, e.g., in the underlying representation formalism, and whether they are equipped with an explicit notion of quality, but most of them rely on reasoning mechanisms to support the design of the ontology. Semantics is a prerequisite for *reasoning support*: derivations such as the above can be made mechanically, instead of being made by hand. Logic-based reasoning is employed by the tools to verify the specification, infer implicit statements and facts, and manifest any inconsistencies. Reasoning support is important because it allows one to

– check the consistency of the ontology and the knowledge;
– check for unintended relationships between classes;
– derive explicitly all the statements that are true in the ontology, to better understand its properties;
– reduce the redundancy of an ontology, discover equivalent descriptions, reuse concept descriptions, and refine the definitions;
– automatically classify instances in classes.

In addition to the so called *standard* reasoning support listed above, non-standard inference for ontologies are of great practical impact in ontology-based applications. In particular, tools for building and maintaining large knowledge bases also requires system services that cannot be provided by the standard reasoning techniques. These non-standard reasoning problems encompass matching and unification of concepts (useful, e.g., for browsing ontologies and detecting redundancies), least-common-subsumer and most-specific-concept computation (useful to support the definition of new concepts), and approximation of concepts (useful for approximate reasoning and for a comprehensible presentation of ontologies to non-expert users).

Automated reasoning support allows one to check many more cases than what can be done manually. Checks like the above are valuable for

– *designing* large ontologies, where multiple authors are involved;
– *integrating and sharing* ontologies from various sources.

Formal semantics and reasoning support is usually provided by mapping an ontology language to a known logical formalism, and by using automated reasoners that already exist for those formalisms.

3 The Key Semantic Web Ontology Languages

We now turn to a discussion of specific ontology languages that are based on the abstract view from the previous version: RDF Schema and OWL. Quite a few other sources already exist that give general introductions to these languages. Some parts of the RDF and OWL specifications are intended as such introductions (in particular [13], [9] and [10]), and also didactic material such as [12] and [11].

Our presentation is structured along the so-called layering of OWL: OWL Lite, OWL DL and OWL Full. This layering is motivated by different requirements that different users have for a Web ontology language:

– RDF(S) is intended for those users primarily needing a classification hierarchy with typing of properties and meta-modelling facilities;
– OWL Lite adds the possibility to express definitions and axioms, together with a limited use of properties to define classes;
– OWL DL supports those users who want the maximum expressiveness while retaining good computational properties;
– OWL Full is meant for users who want maximum expressiveness with no computational guarantees.

Before discussing the language primitives of OWL Lite, we first discuss language elements from RDF and RDF Schema (RDF(S) for short). With the only purpose to simplify the presentation in this tutorial by obtaining a strict layering between RDF(S) and OWL Lite, we will restrict our discussion of RDF(S) to the case where the vocabulary is strictly partitioned, the meta-modelling and reification facilities are forbidden, as described in [12], also called “type separation” in [9]:

> "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. This means that, for example, a class cannot at the same time be an individual, [...]"

Under this restriction, we have the following strict language inclusion relationship:

$$\text{RDF(S)} \subset \text{OWL Lite} \subset \text{OWL DL},$$

where $\subset$ stands for both syntactic and semantic language inclusion, in other words: every syntactically correct RDF(S) statement is also a correct OWL Lite statement, and every model of a RDF(S) ontology is also a model for the same ontology expressed in OWL Lite (and similarly for the other case). A similar but less strong restriction was proposed with RDFS(FA) [7], which does allow a class to be an instance of another class, as long as this is done in a stratified fashion. When dropping the restriction of a partitioned or stratified vocabulary for RDF(S), the first inclusion relationship no longer holds. In that case, RDF(S) is only a sublanguage of OWL Full. However, note that even in the general case when the inclusion does not hold RDF(S) and OWL Lite/DL can still easily inter-operate. Also note that the inclusion between OWL DL and OWL Full does not hold, intuitively due to the lack of reification in OWL DL and OWL Lite.

Before we discuss the different language primitives that we encounter along this set of inclusions, we first list some of our notational conventions.

We use the normative abstract syntax for OWL as defined in [15]. While this syntax in only meant for OWL itself, we use the same syntax for introducing RDF(S) in order to clarify the relation between the languages[1]. We will use symbols c_i for classes, e_i for objects, p_i for properties between objects, and o_i for ontologies. Whenever useful, we will prefix classes and instances with pseudo-namespaces to indicate the ontology in which these symbols occur, e.g. $o_1\, e_1$ and $o_2\, e_1$ are two different instances, the first occurring in ontology o_1, the second in ontology o_2.

Note that the XML-based syntax is far better known, but arguably not as readable. In fact, the XML-syntax is clearly geared towards machine processing, while the abstract syntax is tailored to human reading, thus our choice in this section. The reader should keep in mind that the characteristics of the ontology languages are independent of the syntax used.

3.1 RDF Schema

The most elementary building block of RDF(S) is a class, which defines a group of individuals that belong together because they share some properties. The following states that an instance e belongs to a class c:

`Individual(e type(c))` ("e is of type c").

[1] Note that the semantics of the same constructs in RDF(S) and OWL can differ.

The second elementary statement of RDF(S) is the subsumption relation between classes: `subClassOf`:

$$\texttt{subClassOf}(\mathtt{c_i}\ \mathtt{c_j})$$

In RDF, instances are related to other instances through properties:

$$\texttt{Individual}(\mathtt{e_i}\ \texttt{value}(\mathtt{p}\ \mathtt{e_j}))$$

Properties are characterised by their domain and range:

$$\texttt{ObjectProperty}(\mathtt{p}\ \texttt{domain}(\mathtt{c_i})\texttt{range}(\mathtt{c_j}))$$

Finally, just as with classes, properties are organised in a subsumption hierarchy:

$$\texttt{SubPropertyOf}(\mathtt{o_1} : \mathtt{p_i}\ \mathtt{o_2} : \mathtt{p_j})$$

RDF and RDFS allow the representation of *some* ontological knowledge. The main modelling primitives of RDF/RDFS concern the organisation of vocabularies in typed hierarchies: subclass and subproperty relationships, domain and range restrictions, and instances of classes. However a number of other features are missing. Here we list a few:

– *Local scope of properties:* `rdfs:domain` and `fs:range` define a unique domain/range of a property for all classes. Thus in RDF Schema we cannot declare domain/range restrictions that apply to some classes only. For example, for the property "father of", the father of elephants are elephants, while the fathers of mice are mice.
– *Disjointness of classes:* Sometimes we wish to say that classes are disjoint. For example, `male` and `female` are disjoint. But in RDF Schema we can only state subclass relationships, e.g. `female` is a subclass of `person`.
– *Boolean combinations of classes:* Sometimes we wish to build new classes by combining other classes using union, intersection and complement. For example, we may wish to define the class `person` to be the disjoint union of the classes `male` and `female`. RDF Schema does not allow such definitions.
– *Cardinality restrictions:* Sometimes we wish to place restrictions on how many distinct values a property may take. For example, we would like to say that a car has at most four wheels. Again such restrictions are impossible to express in RDF Schema. Note that min cardinality restrictions can be expressed *for individuals* in RDF(S) by making use of the b-nodes.
– *Special characteristics of properties:* Sometimes it is useful to say that a property is *transitive* (like "greater than"), *unique* (like "has mother"), or the *inverse* of another property (like "eats" and "is eaten by").

Summary of Basic Features of RDF Schema.

– Classes and their instances
– Binary properties between objects
– Organisation of classes and properties in hierarchies
– Types for properties: domain and range restrictions

4 Web Ontology Language OWL

4.1 OWL Lite

One of the significant limitations of RDF Schema is the inability to make equality claims between individuals. Such equality claims are possible in OWL Lite:

$$\texttt{SameIndividual}(\mathtt{e_i}\ \mathtt{e_j})$$

Besides equality between instances, OWL Lite also introduces constructions to state equality between classes and between properties. Although such equalities could already be expressed in an indirect way in RDF(S) (e.g., through a pair of mutual `Subclassof` or `SubPropertyOf` statements), this can be done directly in OWL Lite:

$$\texttt{EquivalentClasses}(\mathtt{c_1}\ \mathtt{c_j})$$
$$\texttt{EquivalentProperties}(\mathtt{p_1}\ \mathtt{p_j})$$

Just as importantly, as making positive claims about equality or subsumption relationships, is stating negative information about inequalities. A significant limitation of RDF(S)[2] is the inability to state such inequalities. Since OWL does not make the unique name assumption, two instances e_i and e_j are not automatically regarded as different. Such an inequality must be explicitly stated, as:

$$\texttt{DifferentIndividuals}(\mathtt{e_i}\ \mathtt{e_j})$$

Because inequality between individuals is an often occurring and important statement (in many ontologies, all differently named individuals are assumed to be different, i.e. they embrace the unique name assumption), OWL Lite provides an abbreviated form:

$$\texttt{DifferentIndividuals}(\mathtt{e_1}\ ...\ \mathtt{e_4})$$

abbreviates the six `DifferentIndividuals` statements that would have been required for this.

Whereas the above constructions are aimed at instances and classes, OWL Lite also has constructs specifically aimed at properties. An often occurring phenomenon is that a property can be modelled in two directions. Examples are *ownerOf* vs. *ownedBy*, *contains* vs. *isContainedIn*, *childOf* vs. *parentOf* and countless others. The relationship between such pairs of properties is established by stating

$$\texttt{ObjectProperty}(\mathtt{p_i}\ \texttt{inverseOf}(\mathtt{p_j}))$$

Other vocabulary in OWL Lite (`TransitiveProperty` and `SymmetricProperty` are modifying a single property, rather then establishing a relation between two properties:

[2] But motivated by a deliberate design decision concerning the computational and conceptual complexity of the language.

$$\mathtt{ObjectProperty}(\mathrm{o_1:p_i}\ \mathtt{Transitive})$$
$$\mathtt{ObjectProperty}(\mathrm{o_1:p_i}\ \mathtt{Symmetric})$$

The main limitation of RDF(S) to represent knowledge in terms of concepts and their properties, is its inability to use properties in the local context of a class. As we have already noted, a property has a unique definition for its domain and for its range, and moreover the participation constraints of the instances of the domain and range classes to the property are not specifiable in RDF(S). So, in RDF(S) it is impossible to state whether a property is optional or required for the instances of the class (in other words: should it have at least one value or not), and whether it is single- or multi-valued (in other words: is it allowed to have more than one value or not). Technically, these restrictions constitute 0/1-cardinality constraints on the property. The case where a property is allowed to have at most one value for a given instance (i.e. a max-cardinality of 1) has a special name: `FunctionalProperty`. The case where the value of a property uniquely identifies the instance of which it is a value (i.e. the inverse property has a max-cardinality of 1) is called `InverseFunctionalProperty`. These two constructions allow for some interesting derivations under the OWL semantics: If an ontology models that any object can only have a single "age":

$$(\mathtt{ObjectProperty}\ \mathrm{age}\ \mathtt{Functional})$$

then different age-values for two instances e_i and e_j allow us to infer that

$$\mathtt{DifferentIndividuals}(\mathrm{e_i}\ \mathrm{e_j})$$

(if two objects e_i and e_j have a different age, they must be different objects). Similarly, if an ontology states that social security numbers uniquely identify individuals, i.e.

$$\mathtt{ObjectProperty}(\mathrm{hasSSN}\ \mathtt{InverseFunctional})$$

then the two facts

$$\mathtt{Individual}(\mathrm{e_i}\ \mathtt{value}(\mathrm{hasSSN}\ 12345))$$
$$\mathtt{Individual}(\mathrm{e_j}\ \mathtt{value}(\mathrm{hasSSN}\ 12345))$$

sanction the derivation of the fact

$$\mathtt{SameIndividuals}(\mathrm{e_i}\ \mathrm{e_j})$$

Although RDF(S) already allows to state domain and range restrictions, these are very limited. OWL Lite allows more refined version of these, local to the definition of a class:

$$\mathtt{Class}(\mathrm{c_i}\ \mathtt{restriction}(\mathrm{p_i}\ \mathtt{allValuesFrom}(\mathrm{c_j})))$$

says that all p_i-values (if any) *for each member of* c_i must be members of c_j. This differs from the RDF(S) range restriction

$$\mathtt{ObjectProperty}(\mathrm{p}\ \mathtt{range}(\mathrm{c_j}))$$

which says that all p_i-values must be members of c_j, irrespective of whether they are members of c_i or not. This allows us to use the same property-name p_i with different range restrictions c_j depending on the class c_i to which p_i is applied. For example, take for p_i the property `Parent`. Then `Parent`s of cats are cats, while `Parent`s of dogs are dogs. An RDF(S) range restriction would not be able to capture this.

Similarly, although in RDF(S) we can define the range of a property, we cannot enforce that properties actually do have a value: we can state the authors write books:

```
ObjectProperty(write domain(author) range(book))
```

but we cannot enforce in RDF(S) that every author must have written at least one book. This is possible in OWL Lite:

```
Class(author restriction(write someValuesFrom(book)))
```

Technically speaking, these are just special cases of the general cardinality constraints allowed in OWL DL. The `someValuesFrom` corresponds to a min-cardinality constraint with value 1, and the functional property constraint mentioned above can be rewritten in this context with a max-cardinality constraint with value 1. These can also be stated directly:

```
Class(author restriction(write minCardinality(1)))
  Class(object restriction(age maxCardinality(1)))
```

When a property has a `minCardinality` and `maxCardinality` constraints with the same value, these can be summarised by a single exact `Cardinality` constraint.

4.2 OWL DL

With the step from OWL Lite to OWL DL, we obtain a number of additional language constructs, which simplify the writing of an ontology, even if most of them could be written anyway in OWL Lite as macros. It is often useful to say that two classes are disjoint (which is much stronger than saying they are merely not equal):

$$\texttt{DisjointClasses}(\texttt{c}_\texttt{i}\ \texttt{c}_\texttt{j})$$

OWL DL allows arbitrary Boolean algebraic expressions on either side of an equality of subsumption relation. For example

$$\texttt{SubClassOf}(\texttt{c}_\texttt{i}\ \texttt{unionOf}(\texttt{c}_\texttt{j}\ \texttt{c}_\texttt{k}))$$

In other words: c_i is not subsumed by either c_j or c_k, but is subsumed by their union. Similarly

$$\texttt{EquivalentClasses}(\texttt{c}_\texttt{i}\ \texttt{intersectionOf}(\texttt{c}_\texttt{j}\texttt{c}_\texttt{k}))$$

in other words: although c_i is subsumed by c_j and c_k (a statement already expressible in RDF(S)), stating that c_i is equivalent to their intersection is much stronger. An obvious example to think of here is “old men”: “old men” are not just both old and men, but they are *exactly* the intersection of these two properties.

Of course, the `unionOf` and `intersectionOf` may be taken over more than two classes, and may occur in arbitrary Boolean combinations.

Besides disjunction (`unionOf`) and conjunction (`intersectionOf`), OWL DL completes the Boolean algebra by providing a construct for negation: `complementOf`:

$$\texttt{complementOf}(\texttt{c}_\texttt{i}\ \texttt{c}_\texttt{j})$$

In fact, arbitrary class expressions can be used on either side of subsumption or equivalence axioms.

Note that all the additional OWL DL constructs introduced so far, are also indirectly expressible already in OWL Lite. For example, the disjointness between two classes $\texttt{c}_\texttt{i}$ and $\texttt{c}_\texttt{j}$can be expressed by means of the following two statements in OWL Lite, for some fresh new property `p`:

$$\texttt{SubClassOf}(\texttt{c}_\texttt{i}\ \texttt{restriction}(\texttt{p minCardinality}(1)))$$
$$\texttt{SubClassOf}(\texttt{c}_\texttt{j}\ \texttt{restriction}(\texttt{p maxCardinality}(0)))$$

There are cases where it is not possible to define a class in terms of such algebraic expressions. This can be either impossible in principle. In such cases it is sometimes useful to simply enumerate sets of individuals to define a class. This is done in OWL DL with the `oneOf` construct:

$$\texttt{EquivalentClasses}(\texttt{c}_\texttt{j}\ \texttt{oneOf}(\texttt{e}_1 \ldots \texttt{e}_\texttt{n}))$$

Similar to defining a class by enumeration, we can define a property to have a specific value by stating the value:

$$\texttt{Class}(\texttt{c}_\texttt{i}\ \texttt{restriction}(\texttt{p}_\texttt{j}\ \texttt{hasValue}\ \texttt{e}_\texttt{k})$$

The extension from OWL Lite to OWL DL also lifts the restriction on cardinality constraints to have only 0/1 values.

4.3 OWL Full

OWL Lite and DL are based on a strict segmentation of the vocabulary: no term can be both an instance and a class, or a class and a property, etc. Full RDF(S) is much more liberal: a class c_1 can have both a `type` and a `subClassOf` relation to a class c_2, and a class can even be an instance of itself. In fact, the class `Class` is a member of itself. OWL Full inherits from RDF(S) this liberal approach. This feature is crucial for using OWL as a meta-modelling language.

Schreiber [14] argues that this is exactly what is needed in many cases of practical ontology integration. When integrating two ontologies, opposite commitments have often been made in the two ontologies on whether something is modelled as a class or an instance. This is less unlikely than it may sound: is "747" an *instance* of the class of all airplane-types made by Boeing or is "747" a *subclass* of the class of all airplanes made by Boeing, and are particular jet planes instances of this subclass? Both points of view are defensible. In OWL Full, it is possible to have equality statements between a class and an instance.

In fact, just as in RDF Schema, OWL Full allows us even to apply the constructions of the language to themselves. It is perfectly legal to (say) apply a max-cardinality constraint of 2 on the subClassOf relationship. For this reason, OWL Full does not include OWL DL, in which the constructions of the language are not semantic objects. Of course, building any complete and terminating reasoning tools that support this very liberal self-application of the language is out of the question. In fact, the theory shows that it is impossible to build a correct and complete inference engine for OWL Full.

5 Other Web-Based Ontology Languages

Besides the two standards RDF Schema and OWL discussed above, a number of other approaches for encoding ontologies on the World Wide Web have been proposed in the past. A comparison of these older languages is reported in [16]. We will now briefly review the results of this comparison and discuss implications for our work.

Besides RDF Schema and OWL[3], which have been introduced above, the comparison reported in [16] includes the following languages that have been selected on the basis of their aim of supporting knowledge representation on the Web and their compatibility to the Web standards XML or RDF.

- *XOL (XML-based ontology language).* XOL [4] has been proposed as a language for exchanging formal knowledge models in the domain of bioinformatics. The development of XOL has been guided by the representational needs of the domain and by existing frame-based knowledge representation languages.
- *SHOE (simple HTML ontology extension).* SHOE[6] was created as an extension of HTML for the purpose of defining machine-readable semantic knowledge. The aim of SHOE is to enable intelligent Web agents to retrieve and gather knowledge more precisely than it is possible in the presence of plain HTML documents.
- *OML: (ontology markup language).* OML [5] is an ontology language that has initially been developed as an XML serialisation of SHOE. Meanwhile, the language consists of different layers with increasing expressiveness. The semantics especially of the higher levels is largely based on the notion of conceptual graphs. In the comparison, however, only a less expressive subset of OML (simple OML) is considered.

[3] Actually, [16] discuss DAML+OIL instead of OWL. DAML+OIL [8] is the direct precursor of OWL, and all of the conclusions from [16] about DAML+OIL are also valid for OWL.

Table 1. Comparison of web ontology languages with respect to concepts and taxonomies (taken from [16])

	XOL	**SHOE**	**OML**	**RDF/S**	**OIL**	**DAML+OIL**
Partitions	−	−	+	−	+	+
Attributes						
Instance attr.	+	+	+	+	+	+
Class attr.	+	−	+	−	+	+
Local scope	+	+	+	+	+	+
Global scope	+	−	+	+	+	+
Facets						
Default values	+	−	−	−	−	−
Type constr.	+	+	+	+	+	+
Cardinalities	+	−	−	−	+	+
Taxonomies						
Subclass of	+	+	+	+	+	+
Exhaustive comp.	−	−	+	−	+	+
Disjoint comp.	−	−	+	−	+	+
Not subclass of	−	−	−	−	+	+

– *OIL (ontology inference layer).* OIL [3] is an attempt to develop an ontology language for the Web that has a well defined semantics and sophisticated reasoning support for ontology development and use. The language is constructed in a layered way starting with core-OIL, providing a formal semantics for RDF Schema, standard-OIL, which is equivalent to an expressive description logic with reasoning support, and Instance OIL that adds the possibility of defining instances.

We have to mention that there is a strong relationship between the OIL language and RDF Schema as well as DAML+OIL. OIL extends RDF Schema and has been the main influence in the development if DAML+OIL. The main difference between OIL and DAML+OIL is an extended expressiveness of DAML+OIL in terms of complex definitions of individuals and data types. DAML+OIL in turn has been the basis for the development of OWL, which carries the stamp of an official W3C recommendation. All observations on DAML+OIL in this comparison also apply to OWL.

6 Description Logics

We briefly now introduce description logics, which is the logic-based formalism which is behind the OWL family of web ontology languages. From this brief Section the parallel with the OWL family of web ontology languages will appear clear. An extensive treatment of description logics, from friendly introductory chapters, to the theoretical results, up to the description of applications and systems, can be found in the Handbook of Description Logics [1]. Consistently with the informal notion of semantics introduced above for the web ontology languages, description logics are considered as a *structured* fragment of predicate logic. $\mathcal{ALC}$ is the minimal description language including full negation and disjunction—i.e., propositional calculus.

The basic types of a DL language are *concepts*, *roles*, and *features*. A concept is a description gathering the common properties among a collection of individuals; from a logical point of view it is a unary predicate ranging over the domain of individuals. A concept corresponds to a class in the web ontology languages. Inter-relationships between these individuals are represented either by means of roles (which are interpreted as binary relations over the domain of individuals) or by means of features (which are interpreted as partial functions over the domain of individuals). Roles correspond to properties of RDF and OWL, while features correspond to functional properties. In this Section, we will consider the Description Logic $\mathcal{ALCQI}$, extending $\mathcal{ALC}$ with qualified cardinality restrictions and inverse roles.

According to the syntax rules of Figure 1, $\mathcal{ALCQI}$ *concepts* (denoted by the letters C and D) are built out of *primitive concepts* (denoted by the letter A), *roles* (denoted by the letter R), and *primitive features* (denoted by the letter f); roles are built out of *primitive roles* (denoted by the letter P) and *primitive features.* The top part of Figure 1 defines the $\mathcal{ALC}$ sublanguage. Please also note that features are introduced as shortcuts; in fact, they can be expressed by means of axioms using cardinality restrictions, as we already noticed for OWL DL.

$C, D \rightarrow$	$A \mid$	A	(primitive conc.)
	$\top \mid$	`top`	(top)
	$\bot \mid$	`bottom`	(bottom)
	$\neg C \mid$	(`not` C)	(complement)
	$C \sqcap D \mid$	(`and` C D ...)	(conjunction)
	$C \sqcup D \mid$	(`or` C D ...)	(disjunction)
	$\forall R. C \mid$	(`all` R C)	(univ. quantifier)
	$\exists R. C \mid$	(`some` R C)	(exist. quantifier)
	$f\uparrow \mid$	(`undefined` f)	(undefinedness)
	$f : C \mid$	(`in` f C)	(selection)
	$\geq n\, R. C \mid$	(`atleast` n R C)	(min cardinality)
	$\leq n\, R. C$	(`atmost` n R C)	(max cardinality)
$R \rightarrow$	$P \mid$	P	(primitive role)
	$f \mid$	f	(primitive feature)
	R^{-1}	(`inverse` R)	(inverse role)

Fig. 1. Syntax rules for $\mathcal{ALCQI}$

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}\\
\bot^{\mathcal{I}} &= \emptyset\\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}\\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}\\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}\\
(\forall R.\, C)^{\mathcal{I}} &= \{i \in \Delta^{\mathcal{I}} \mid \forall j.\, R^{\mathcal{I}}(i,j) \Rightarrow C^{\mathcal{I}}(j)\}\\
(\exists R.\, C)^{\mathcal{I}} &= \{i \in \Delta^{\mathcal{I}} \mid \exists j.\, R^{\mathcal{I}}(i,j) \wedge C^{\mathcal{I}}(j)\}\\
(f\!\uparrow)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus \operatorname{dom} f^{\mathcal{I}}\\
(f : C)^{\mathcal{I}} &= \{i \in \operatorname{dom} f^{\mathcal{I}} \mid C^{\mathcal{I}}(f^{\mathcal{I}}(i))\}\\
(\geq n\, R.\, C)^{\mathcal{I}} &= \{i \in \Delta^{\mathcal{I}} \mid\ \sharp\{j \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(i,j) \wedge C^{\mathcal{I}}(j)\} \geq n\}\\
(\leq n\, R.\, C)^{\mathcal{I}} &= \{i \in \Delta^{\mathcal{I}} \mid\ \sharp\{j \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(i,j) \wedge C^{\mathcal{I}}(j)\} \leq n\}\\
(R^{-1})^{\mathcal{I}} &= \{(i,j) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(j,i)\}
\end{aligned}
$$

Fig. 2. Extensional semantics of $\mathcal{ALCQI}$

Let us now consider the formal semantics of $\mathcal{ALCQI}$. We define the *meaning* of concepts as sets of individuals—as for unary predicates—and the meaning of roles as sets of pairs of individuals—as for binary predicates. This is the formalised notion of instantiation of the domain we introduced at the beginning of this chapter. Formally, an *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a set $\Delta^{\mathcal{I}}$ of individuals (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *interpretation function* of $\mathcal{I}$) mapping every concept to a subset of $\Delta^{\mathcal{I}}$, every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every feature to a partial function from $\Delta^{\mathcal{I}}$ to $\Delta^{\mathcal{I}}$, such that the equations in Figure 2 are satisfied. The semantics of the language can also be given by stating equivalences among expressions of the language and First Order Logic formulae. An atomic concept A, an atomic role P, and an atomic feature f, are mapped respectively to the open formulæ$A(\gamma)$, $P(\alpha,\beta)$, and $f(\alpha,\beta)$ – with f a functional relation, also written $f(\alpha) = \beta$. Figure 3 gives the transformational semantics of $\mathcal{ALCQI}$ expressions in terms of equivalent FOL well-formed formulæ. A concept C and a role R correspond to the FOL open formulae $F_C(\gamma)$ and $F_R(\alpha,\beta)$ respectively. It is worth noting that, using the standard model-theoretic semantics, the extensional semantics of Figure 2 can be derived from the transformational semantics of Figure 3.

For example, we can consider the concept of HAPPY FATHERS, defined using the primitive concepts `Man, Doctor, Rich, Famous` and the roles `CHILD, FRIEND`. The concept HAPPY FATHERS can be expressed in $\mathcal{ALCQI}$ as

$$
\begin{aligned}
&\texttt{Man} \sqcap (\exists\texttt{CHILD}.\, \top)\sqcap\\
&\forall\texttt{CHILD}.\, (\texttt{Doctor} \sqcap \exists\texttt{FRIEND}.\, (\texttt{Rich} \sqcup \texttt{Famous})),
\end{aligned}
$$

i.e., those men having some child and all of whose children are doctors having some friend who is rich or famous.

An ontology is called in DL a *knowledge base*, and formally it is a finite set Σ of *terminological axioms* – these are the ontology statements; it can also be called a *terminology* or TBox. For a concept name A, and (possibly complex)

$$
\begin{aligned}
\top^{\mathcal{I}} &\sim \mathsf{true}\\
\bot^{\mathcal{I}} &\sim \mathsf{false}\\
(\neg C)^{\mathcal{I}} &\sim \neg F_C(\gamma)\\
(C \sqcap D)^{\mathcal{I}} &\sim F_C(\gamma) \wedge F_D(\gamma)\\
(C \sqcup D)^{\mathcal{I}} &\sim F_C(\gamma) \vee F_D(\gamma)\\
(\exists R.\, C)^{\mathcal{I}} &\sim \exists x.\, F_R(\gamma, x) \wedge F_C(x)\\
(\forall R.\, C)^{\mathcal{I}} &\sim \forall x.\, F_R(\gamma, x) \Rightarrow F_C(x)\\
(f\!\uparrow)^{\mathcal{I}} &\sim \neg\exists x.\, f(\gamma, x)\\
(f : C)^{\mathcal{I}} &\sim \exists x.\, f(\gamma, x) \wedge F_C(x)\\
(\geq n\, R.\, C)^{\mathcal{I}} &\sim \exists^{\geq n} x.\, F_R(\gamma, x) \wedge F_C(x)\\
(\leq n\, R.\, C)^{\mathcal{I}} &\sim \exists^{\leq n} x.\, F_R(\gamma, x) \wedge F_C(x)\\
(R^{-1})^{\mathcal{I}} &\sim F_R(\beta, \alpha)
\end{aligned}
$$

Fig. 3. FOL semantics of $\mathcal{ALCQI}$

concepts C, D, terminological axioms are of the form $A \doteq C$ (concept definition), $A \sqsubseteq C$ (primitive concept definition), $C \sqsubseteq D$ (general inclusion statement). An interpretation $\mathcal{I}$ satisfies $C \sqsubseteq D$ if and only if the interpretation of C is included in the interpretation of D, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. It is clear that the last kind of axiom is a generalisation of the first two: concept definitions of the type $A \doteq C$ – where A is an atomic concept – can be reduced to the pair of axioms $(A \sqsubseteq C)$ and $(C \sqsubseteq A)$. Another class of terminological axioms – pertaining to roles R, S – are of the form $R \sqsubseteq S$. Again, an interpretation $\mathcal{I}$ satisfies $R \sqsubseteq S$ if and only if the interpretation of R – which is now a set of *pairs* of individuals – is included in the interpretation of S, i.e., $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a *model* of a knowledge base Σ iff every terminological axiom of Σ is satisfied by $\mathcal{I}$. If Σ has a model, then it is *satisfiable*; thus, checking for KB satisfiability is deciding whether there is at least one model for the knowledge base. Σ *logically implies* an axiom α (written $\Sigma \models \alpha$) if α is satisfied by every model of Σ. We say that a concept C is *subsumed* by a concept D in a knowledge base Σ (written $\Sigma \models C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of Σ. For example, the concept

$\mathtt{Person} \sqcap (\exists \mathtt{CHILD}.\, \mathtt{Person})$

denoting the class of PARENTS—i.e., the persons having at least a child which is a person—subsumes the concept

$\mathtt{Man} \sqcap (\exists \mathtt{CHILD}.\, \top) \sqcap$
$\forall \mathtt{CHILD}.\, (\mathtt{Doctor} \sqcap \exists \mathtt{FRIEND}.\, (\mathtt{Rich} \sqcup \mathtt{Famous}))$

denoting the class of HAPPY FATHERS – with respect to the following knowledge base Σ:

$\mathtt{Doctor} \doteq \mathtt{Person} \sqcap \exists \mathtt{DEGREE}.\, \mathtt{Phd}$,
$\mathtt{Man} \doteq \mathtt{Person} \sqcap \mathtt{sex} : \mathtt{Male}$,

i.e., every happy father is also a person having at least one child, given the background knowledge that men are male persons, and that doctors are persons.

A concept C is satisfiable, given a knowledge base Σ, if there is at least one model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}} \neq \emptyset$, i.e. $\Sigma \not\models C \equiv \bot$. For example, the concept

$$(\exists \texttt{CHILD.Man}) \sqcap (\forall \texttt{CHILD.}(\texttt{sex} : \neg\texttt{Male}))$$

is unsatisfiable with respect to the above knowledge base Σ. In fact, an individual whose children are not male cannot have a child being a man.

7 The Importance of Correct Inference

An ontology inference engine based on description logics (such as iFaCT or Racer) can offer a reasoning service to applications willing to properly use an ontology. As we have already noticed, the inferential process's complexity depends strictly on the adopted ontology language's expressivity: the inference engine becomes increasingly complex as the ontology language becomes more expressive. In fact, theoreticians have proved that you can't build a complete inference engine for OWL Full, although it's possible to use existing description logic systems as inference engines for OWL Lite and OWL DL.

Designing and implementing complete inference engines for expressive ontology languages isn't easy. As a prerequisite, you must have formal proof that the algorithms are complete with respect to the ontology language's declared semantics. The description logics community – which provides the theoretical foundations to the OWL family of web ontology languages – has 20-plus years of experience to help provide theoretical results, algorithms, and efficient inference systems for all but the most expressive OWL languages. We can understand how important it is for an inference engine to be complete with the following example.

Suppose a military agency asks you to write an ontology to recognise whether a particular individual description indicates some sort of "enemy" concept so that an application can take appropriate automatic action (such as shooting) given the inference engine's answer. If the inference engine is sound but incomplete, it will recognise most but not all enemies because it isn't a complete reasoner. Because it is sound, however, it won't confuse a friendly soldier with an enemy. So, the application will start the automatic shooting procedure only when the system recognises without doubt that someone is an enemy. The application could fail to shoot an enemy, but field soldiers can take traditional backup (nonautomatic) action. Soundness is more important because you don't want to shoot your own soldiers. So far, so good.

The agency has another application strictly related to the first one. The task is now to recognise an individual description as an allied soldier to activate automatic procedures that will alert the soldier to the headquarters' secret position. Again, the system must have a sound inference engine because the agency doesn't want to disclose secret information to enemies. Moreover, incompleteness is not a major problem because the defence system can still be valid even if a soldier doesn't know where the headquarters is located.

The agency decides, of course, to use the same shared ontology for both applications. After all, the task in one case is to decide whether a soldier is an enemy and in the other case decide whether he or she isn't. So the second application can use the same ontology as the first, but it exploits the outcome in a dual way. Unfortunately, it turns out that the agency can't use the same ontology for both tasks if the ontology language's inference engine is sound but incomplete. If a sound but incomplete reasoning system exists for solving, say, the first problem (recognising enemies), you can't use the same reasoning system as a sound (and possibly incomplete) procedure for solving the second problem (recognising allies). In fact, using the same procedure for solving the second problem would be unsound – it will say an individual isn't an enemy when he or she actually is. Although this is harmless for the first problem, it is bad for the second, dual one. It would disclose valuable military secrets to enemies.

To solve this problem, one must have both a sound and complete inference engine for the ontology language. This rules out using OWL Full for the above application because having a complete inference engine with this language is impossible. The same of course holds for OWL DL inference engines without guaranteed completeness properties.

It is important that Semantic Web application developers consider properly whether such completeness properties are required for their applications.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.
2. F. van Harmelen and D. Fensel. Practical Knowledge Representation for the Web. In *Proc. IJCAI'99 Workshop on Intelligent Information Integration*, 1999
3. D. Fensel, I. Horrocks, F. van Harmelen, D.L. McGuinness and Peter F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems* 16,2 (2001): 38-44
4. P. Karp, V. Chaudri and J. Thomere. An XML-Based Ontology Exchange Language. Available at http://www.ai.sri.com/∼ pkarp/xol
5. R. Kent. Conceptual Knowledge Modelling Language. Available at http://www.ontologos.org/CKML/
6. S. Luke and J. Hefflin. SHOE 1.01 Proposal Specification. Available at http://www.cs.umd.edu/projects/plus/SHOE
7. J. Pan and I. Horrocks. (FA) and RDF MT: Two Semantics for RDFS. In *Proc. 2003 International Semantic Web Conference (ISWC 2003)*, LNCS 2870, Springer 2003,30-46
8. P. Patel-Schneider, I. Horrocks and F. van Harmelen. Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web. In *Proc. Eighteenth National Conference on Artificial Intelligence*, AAAI Pres 2002
9. D.L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Available at http://www.w3.org/TR/owl-features/
10. M.K. Smith, Chris Welty and D.L. McGuinness. OWL Web Ontology Language Guide. Available at http://www.w3.org/TR/owl-guide/
11. G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. In S. Staab and R. Studer (Eds), *Handbook on Ontologies in Information Systems*, Springer 2003

12. G. Antoniou and F. van Harmelen. *A Semantic Web Primer*, MIT Press 2004
13. F. Manola and E. Miller. RDF Primer. Available at http://www.w3c.or.kr/Translation/PR-rdf-primer-20031215/
14. G. Schreiber. The Web is not well-formed. *IEEE Intelligent Systems* 17,2 (2002)
15. P.F. Patel-Schneider, P. Hayes and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. Available at http://www.w3.org/TR/owl-semantics/
16. A. Gomez-Perez and O. Corcho. Ontology Languages for the Semantic Web. *IEEE Intelligent Systems* 2002, 54-60