From SHIQ and RDF to OWL: the making of a Web Ontology Language

Ian Horrocks [a,*], Peter F. Patel-Schneider [b], Frank van Harmelen [c]

[a] *Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK*
[b] *Bell Labs Research, Lucent Technologies, 600 Mountain Avenue, Murray Hill, NJ 07974, USA*
[c] *AI Department, Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081HV Amsterdam, The Netherlands*

Abstract

The OWL Web Ontology Language is a new formal language for representing ontologies in the Semantic Web. OWL has features from several families of representation languages, including primarily Description Logics and frames. OWL also shares many characteristics with RDF, the W3C base of the Semantic Web. In this paper, we discuss how the philosophy and features of OWL can be traced back to these older formalisms, with modifications driven by several other constraints on OWL. Several interesting problems have arisen where these influences on OWL have clashed.
© 2003 Elsevier B.V. All rights reserved.

1. Introduction

OWL [1] is a new ontology language for the Semantic Web, developed by the World Wide Web Consortium (W3C) Web Ontology Working Group. OWL was primarily designed to represent information about categories of objects and how objects are interrelated—the sort of information that is often called an ontology. OWL can also represent information about the objects themselves—the sort of information that is often thought of as data.

OWL was not designed in a vacuum. There were many influences on OWL's design, some mandated by the charter of the Web Ontology Working Group.[1] As OWL is an effort in W3C's Semantic Web activity, it had to fit into the Semantic Web vision of a stack of languages including XML and RDF. As OWL is supposed to be an ontology language, it had to be able to represent a useful group of ontology features. As there were already several ontology languages designed for use in the Web, OWL had to maintain as much compatibility as possible with these existing languages, including SHOE [2], OIL [3] and DAML + OIL [4].

The multiple influences on OWL resulted in some difficult trade-offs. Also, and somewhat surprisingly,

* Corresponding author. Tel.: +44-161-2756133;
fax: +44-161-2756204.
E-mail addresses: horrocks@cs.man.ac.uk (I. Horrocks),
pfps@research.bell-labs.com (P.F. Patel-Schneider),
frank.van.harmelen@cs.vu.nl (F. van Harmelen).

[1] http://www.w3.org/2001/sw/WebOnt/charter.

considerable technical work had to be performed to devise OWL in such a way that it could be shown to have various desirable features, while still retaining sufficient compatibility with its roots. This paper describes some of the trade-offs and design decisions that had to be made by the Web Ontology Working Group during the design of OWL. Although many of these decisions were based on the requirements drawn up for OWL [5], and on solid scientific knowledge, some of them were necessarily based on softer judgements, and some were simply a matter of taste.

This paper presents an account of the trade-offs and design decisions behind OWL. The views presented are those of the authors, and are not necessarily shared by all members of the Web Ontology Working Group.

After a brief introduction and quick survey of OWL, Sections 3 and 4 discuss the historical roots of OWL. Section 5 then surveys some of the major problems that had to be resolved in the design of OWL, while Section 6 describes the solutions and compromises that have been found. Section 7 describes how these solutions have been incorporated in the final design of the OWL language, Section 8 describes extensions of OWL that are already under discussion and Section 9 gives summary of the article.

2. OWL overview

This paper is an account of the design choices and trade-offs that went into the making of OWL, and is *not* meant as an exhaustive description of the OWL language (for which the reader should turn to the OWL documents, including the OWL Language Reference [1] and Guide [6].) Nevertheless, to make the paper self-contained, it contains a short description of the language and its most important uses.

In the context of the Semantic Web, ontologies are expected to play an important role in helping automated processes (so called 'intelligent agents') to access information. In particular, ontologies are expected to be used to provide structured vocabularies that explicate the relationships between different terms, allowing intelligent agents (and humans) to interpret their meaning flexibly yet unambiguously. For example, a suitable pizza ontology might include the in-

formation that Mozzarella and Gorgonzola are kinds of cheese, that cheese is not a kind of meat or fish, and that a vegetarian pizza is one whose toppings do not include any meat or fish. This information allows the term 'pizza topped with (only) Mozzarella and Gorgonzola' to be unambiguously interpreted (by, e.g. a pizza ordering agent) as a specialisation of the term 'vegetarian pizza'.

Terms whose meanings are defined in ontologies can be used in semantic markup that describes the content and functionality of Web-accessible resources [7]. Ontologies and ontology-based semantic markup could be used in

- e-commerce [8], where they can facilitate communication between buying and selling agents by providing a common vocabulary to describe goods (such as pizzas) and services (see, e.g. http://www.verticalnet.com/);
- search engines [9], where they can help in finding pages that contain semantically similar but syntactically different words and phrases (see, e.g. http://www.hotbot.com/); and
- Web and grid services [10,11], where they can provide rich service descriptions that can help in locating suitable services.

In order to support these and other usage scenarios, OWL takes the basic fact-stating ability of RDF [12] and the class- and property-structuring capabilities of RDF schema [13] and extends them in important ways. OWL can declare classes, and organise these classes in a subsumption ('subclass') hierarchy, as can RDF schema. OWL classes can be specified as logical combinations (intersections, unions or complements) of other classes, or as enumerations of specified objects, going beyond the capabilities of RDFS. OWL can also declare properties, organise these properties into a 'subproperty' hierarchy, and provide domains and ranges for these properties, again as in RDFS. The domains of OWL properties are OWL classes, and ranges can be either OWL classes or externally-defined datatypes such as string or integer. OWL can state that a property is transitive, symmetric, functional, or is the inverse of another property, here again extending RDFS.

OWL can express which objects (also called 'individuals') belong to which classes, and what the property values are of specific individuals. Equivalence

statements can be made on classes and on properties, disjointness statements can be made on classes, and equality and inequality can be asserted between individuals.

However, the major extension over RDFS is the ability in OWL to provide restrictions on how properties behave that are local to a class. OWL can define classes where a particular property is restricted so that all the values for the property in instances of the class must belong to a certain class (or datatype); at least one value must come from a certain class (or datatype); there must be at least certain specific values and there must be at least or at most a certain number of distinct values.

For example, using RDFS, we can

- declare classes like `Country`, `Person`, `Student` and `Canadian`;
- state that `Student` is a subclass of `Person`;
- state that `Canada` and `England` are both instances of the class `Country`;
- declare `Nationality` as a property relating to the classes `Person` (its domain) and `Country` (its range);
- state that `age` is a property with `Person` as its domain and integer as its range; and
- state that `Peter` is an instance of the class `Canadian`, and that his age has value 48.

With OWL, we can additionally

- state that `Country` and `Person` are disjoint classes;
- state that `Canada` and `England` are distinct individuals;
- declare `HasCitizen` as the inverse property of `Nationality`;
- state that the class `Stateless` is defined precisely as those members of the class `Person` that have no values for the property `Nationality`;
- state that the class `MultipleNationals` is defined precisely as those members of the class `Person` that have at least two values for the property `Nationality`;
- state that the class `Canadian` is defined precisely as those members of the class `Person` that have `Canada` as a value of the property `Nationality`; and
- state that `age` is a functional property.

The above shows that OWL is quite a sophisticated language. OWL has both an RDF/XML exchange syntax and an abstract frame-like syntax, and it has three named sublanguages. This multiplicity is the direct result of trying to satisfy a large number of sometimes conflicting influences and requirements, as will be discussed throughout the remainder of this paper.

3. Influences on OWL

As mentioned above, the design of OWL has been subject to a variety of influences. These included influences from established formalisms and knowledge representation paradigms, influences from existing ontology languages and influences from existing Semantic Web languages.

Some of the most important influences on the design of OWL came, via its predecessor DAML + OIL, from Description Logics, from the frames paradigm and from RDF. In particular, the formal specification of the language was influenced by Description Logics, the surface structure of the language (as seen in the abstract syntax) was influenced by the frames paradigm and the RDF/XML exchange syntax was influenced by a requirement for upwards compatibility with RDF.

Each of these influences will be examined in more detail in the following sections.

3.1. Description Logics

Description Logics are a family of class-based (concept-based) knowledge representation formalisms [14]. They are characterised by the use of various constructors to build complex classes from simpler ones, an emphasis on the decidability of key reasoning problems, and by the provision of sound, complete and (empirically) tractable reasoning services. Description Logics, and insights from Description Logic research, had a strong influence on the design of OWL, particularly on the formalisation of the semantics, the choice of language constructors and the integration of datatypes and data values. In fact, OWL DL and OWL Lite (two of the three species of OWL) can be viewed as expressive Description Logics, with an ontology being equivalent to a Description Logic knowledge base.

3.1.1. Semantics

A key feature of Description Logics is that they are logics, i.e. formal languages with well-defined semantics. The standard technique for specifying the meaning of a Description Logic is via a *model theoretic* semantics, whose purpose is to explicate the relationship between the language syntax and the intended model(s) of the domain. A model consists of a *domain* (often written as $\Delta^{\mathcal{I}}$) and an *interpretation function* (often written as $\cdot^{\mathcal{I}}$), where the domain is a set of objects and the interpretation function is a mapping from individual, class and property names to elements of the domain, subsets of the domain and binary relations on the domain, respectively. So, for an individual John, $John^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, for a class Person, $Person^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and for a property friend, $friend^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

The interpretation function can be extended from class names to complex class descriptions in the obvious way. For example, given two classes Male and Person interpreted as the sets $Male^{\mathcal{I}} = \{w, x, y\}$ and $Person^{\mathcal{I}} = \{x, y, z\}$, then the intersection of Male and Person (i.e. male persons) is interpreted as the intersection of $\{w, x, y\}$ and $\{x, y, z\}$, i.e. (Male and Person)$^{\mathcal{I}} = \{x, y\}$.

Objects in the domain do not in themselves have any meaning, nor does the choice of any particular set of objects that make up the domain—what is important is the relationships between objects and sets of objects. In a given model, for example, an individual i is an instance of a class C, just in case i is interpreted as an element of the interpretation of C (i.e. $i^{\mathcal{I}} \in C^{\mathcal{I}}$), and a class C is a subclass of a class D just in case the interpretation of C is a subset of the interpretation of D (i.e. $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$).

A Description Logic knowledge base consists of a set of axioms asserting, e.g. that one class is a subclass of another, or that an individual is an instance of a particular class. The meaning of these axioms is given by corresponding constraints on models. If, for example, the knowledge base contains an axiom stating that Person is a subclass of Animal (written as Person ⊑ Animal), then in a model of the knowledge base the interpretation of Person must always be a subset of the interpretation of Animal. The meaning of a knowledge base derives from features and relationships that are common to *all possible models*. If, for example, the interpretation of a class must always be the empty set, then that class is said to be *incon-*sistent, while if there are no possible interpretations, the knowledge base itself is said to be inconsistent. If the relationship specified by a given axiom must hold in all interpretations of a knowledge base, then that axiom is said to be *entailed* by the knowledge base, and if one knowledge base entails every axiom in another knowledge base, then the first knowledge base is said to entail the second knowledge base. A knowledge base containing the axiom Person ⊑ Animal, for example, entails that the intersection of Male and Person is also a subclass of Animal. This entailment is quite trivial, but with a language as complex as OWL, checking entailments may, in general, be a very hard task (see Section 6.5).

Like OIL and DAML + OIL, OWL uses a Description Logic style model theory to formalise the meaning of the language. This was recognised as an essential feature in all three languages, as it allows ontologies, and information using vocabulary defined by ontologies, to be shared and exchanged without disputes as to precise meaning. The need for this kind of formality was reinforced by experience with early versions of the RDF and RDFS specification, where a lack of formality soon led to arguments as to the meaning of language constructs such as domain and range constraints [15]. In order to avoid such problems, the meaning of RDF is now also defined in terms of a model theory [16].

Another advantage of formalising the meaning of the language in this way is that automated reasoning techniques can be used to check the consistency of classes and ontologies, and to check entailment relationships. This is crucial if the full power of ontologies is to be exploited by intelligent agents, and the ability to provide such reasoning support was a key design goal for OWL.

3.1.2. Language constructors

The expressive power of a language like OWL is determined by the class (and property) constructors supported, and by the kinds of axioms that can occur in an ontology. Of course increased expressive power inevitably means increased computational complexity for key reasoning problems such as entailment.

The design of OWL was influenced by more than 10 years of Description Logic research, which has mapped out in considerable detail the complexity–tractability landscape for a wide range of constructors

and axioms, and their various combinations [14]. This knowledge allowed the set of constructors and axioms supported by OWL to be carefully chosen so as to balance the expressive requirements of typical applications with a requirement for reliable and efficient reasoning support.

A particular goal of this design process was to ensure that OWL entailment would at least be decidable, i.e. that it would be possible to design an algorithm that could guarantee to determine whether or not one OWL ontology entails another (such an algorithm is often called a decision procedure). The availability of *practical* decision procedures (for entailment), and even implemented systems, was also an important consideration.

A suitable balance between these computational requirements and the expressive requirements identified in [5] was achieved by basing the design of OWL on the $\mathcal{SH}$ family of Description Logics [17]. The constructors and axioms supported by $\mathcal{SH}$ are similar to those described in Section 2, and include the boolean connectives (intersection, union and complement), restrictions on properties, transitive properties and a property hierarchy, i.e. equivalent to the $\mathcal{ALC}$ Description Logic [18] extended with transitive properties and a property hierarchy. The property hierarchy is important for OWL as it is a feature of RDFS, while transitive properties have been identified as an important requirement in many applications [5]. Members of the $\mathcal{SH}$ family include the influential $\mathcal{SHIQ}$ Description Logic [19], which adds inverse properties and generalised cardinality restrictions, and $\mathcal{SHOQ}(\mathbf{D})$ [20], which adds the ability to define a class by enumerating its instances (e.g. the class {Monday, Tuesday, Wednesday, Thursday, Friday}) and support for datatypes and values (e.g. integer and string datatypes, and values such as '35').

3.1.3. Datatypes

As well as dealing with 'abstract' classes such as Person and Animal, many practical applications need to represent and reason about datatypes and values such as integers and strings. The integration of datatypes in the OWL language is again heavily influenced by Description Logic research, which has demonstrated that care is required in order to avoid complexity blowups or even undecidability

being caused by datatypes [21].[2] In the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic, it was shown that this could be achieved by strictly separating the interpretation of datatypes and values from that of classes and individuals: $\mathcal{SHOQ}(\mathbf{D})$ interpretations include an additional interpretation domain for data values $\Delta_{\mathrm{D}}^{\mathcal{I}}$ which is disjoint from the domain of individuals $\Delta^{\mathcal{I}}$. Datatypes, such as integer, are interpreted as a subset of $\Delta_{\mathrm{D}}^{\mathcal{I}}$, and values such as the integer '35' are interpreted as elements of $\Delta_{\mathrm{D}}^{\mathcal{I}}$. The separation is further strengthened by dividing properties into two disjoint sets of abstract and datatype properties. Abstract properties such as brother are interpreted as binary relations on $\Delta^{\mathcal{I}}$ (i.e. subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$), while datatype properties such as age are interpreted as binary relations between $\Delta^{\mathcal{I}}$ and $\Delta_{\mathrm{D}}^{\mathcal{I}}$ (i.e. subsets of $\Delta^{\mathcal{I}} \times \Delta_{\mathrm{D}}^{\mathcal{I}}$).

This design has the advantage that reasoning with datatypes and values can be almost entirely separated from reasoning with classes and individuals—a class-based reasoner simply needs access to a datatype 'oracle' that can answer simple questions with respect to datatypes and values (e.g. "is -5 a non-negative integer?"). Moreover, the language remains decidable if datatype and value reasoning is decidable, i.e. if oracle can guarantee to answer all questions of the relevant kind for supported datatypes. This can easily be achieved for a range of common datatypes such as integers, decimals and strings [21].

As well as these practical considerations, it can also be argued that the separation of classes and datatypes makes sense from a philosophical standpoint as datatypes are already structured by built-in predicates such as greater-than and less-than. From this point of view, it does not make sense to use ontology axioms to add further structure to datatypes or to form 'hybrid' classes such as the class of red integers.

3.2. Frames paradigm

In the Semantic Web context, where users with a wide range of expertise might be expected to create or modify ontologies, readability and general ease of use are important considerations for an ontology language. In the design of OIL, one of the languages

---

[2] Reasoners for undecidable languages may have undesirable characteristics, including poor performance and/or unpredictability.

on which OWL is based, these requirements were addressed by providing a surface syntax based on the frames paradigm. Frames group together information about each class, making ontologies easier to read and understand, particularly for users not familiar with (Description) Logics. The frames paradigm has been used in a number of well-known knowledge representation systems including the Protégé ontology design tool [22] and the OKBC knowledge model [23]. The design of OIL was influenced by XOL [24]—a proposal for an XML syntax for OKBC Lite (a cut down version of the OKBC knowledge model).

In frame-based languages, each class is described by a frame. The frame includes the name of the class, identifies the more general class (or classes) that it specialises, and lists a set of 'slots'. A slot may consist of a property–value pair, or a constraint on the values that can act as slot 'fillers' (in this context, value means either an individual or a data value). This structure was used in the OIL language, with some enrichment of the syntax for specifying classes and slot constraints so as to enable the full power of a Description Logic style language to be captured. In addition, property frames were used to describe properties, e.g. specifying more general properties, range and domain constraints, transitivity and inverse property relationships.

A class frame is semantically equivalent to a Description Logic axiom asserting that the class being described by the frame is a subclass of each of the classes that it specialises and of each of the property restrictions corresponding to the slots. As well as a richer slot syntax, OIL also offered the possibility of asserting that the class being described by the frame was exactly equivalent to the relevant intersection class (i.e. that they were mutually subsuming). A property frame is equivalent to a set of axioms asserting the relevant subproperty relationships, range and domain constraints, etc. OIL was designed so that OIL frames could easily be mapped to equivalent axioms in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic [25].

The formal specification and semantics of OWL are given by an abstract syntax [26] that has been heavily influenced by frames in general and by the design of OIL in particular. In the abstract syntax, axioms are compound constructions that are very like an OIL-style frame. For classes, they consist of the name of the class being described, a *modality* of 'partial' or 'complete' (indicating that the axiom is asserting a subclass or equivalence relationship, respectively), and a sequence of property restrictions and names of more general classes. Similarly, a property axiom specifies the name of the property and its various features.

The frame style of the abstract syntax makes it *much* easier to read (compared to the RDF/XML syntax), and also easier to understand and use. Moreover, abstract syntax axioms have a direct correspondence with Description Logic axioms, and they can also be mapped to a set of RDF triples.

3.3. RDF syntax

The third major influence on the design of OWL was the requirement to maintain the maximum upwards compatibility with existing Web languages, and in particular with RDF [27]. On the face of it, this requirement made good sense as RDF (and in particular RDF schema) already included several of the basic features of a class and property based ontology language, e.g. it allows subclass and subproperty relationships to be asserted. Moreover, the development of RDF preceded that of OWL, and it seemed reasonable to try to appeal to any user community already established by RDF.

It may seem easy to meet this requirement simply by giving OWL an RDF-based syntax. In order to provide maximum upwards compatibility, however, it was also thought necessary to ensure that the semantics of OWL ontologies was also consistent with the semantics of RDF. This proved to be difficult given the greatly increased expressive power provided by OWL. This will be discussed in more detail in Section 5.

4. Predecessors of OWL

OWL was not the first Web-enabled ontology language, and its design was influenced by several pre-existing languages including RDFS, SHOE, OIL, DAML-ONT and DAML + OIL. DAML + OIL in particular was a major influence on OWL, and the charter of the Web Ontology working group explicitly states that the design of OWL should be based on DAML + OIL. DAML + OIL in turn was heavily influenced by the OIL language, with additional influence from work on DAML-ONT and RDFS.

4.1. SHOE

One of the first attempts at defining an ontology language for deployment on the Web was SHOE [28]. SHOE is a frame-based language with an XML syntax that could be safely embedded in existing HTML documents. SHOE used URI references for names, an important innovation (see Section 7) that was subsequently adopted by both DAML-ONT and DAML + OIL. SHOE also placed emphasis on the fact that ontologies would be tightly interlinked and subject to change. Consequently, SHOE included a number of directives which allowed importing of other ontologies, local renaming of imported constants, and stating versioning and compatibility information between ontologies. This line of thinking has influenced the extra-logical vocabulary of OWL that is designed to partially deal with such issues. SHOE was of lesser influence on the syntactic and semantic design of OWL, since it was not based on RDF and did not come with a formal semantics.

4.2. DAML-ONT

In 1999, the DARPA Agent Markup Language (DAML) program[3] was initiated with the aim of providing the foundations of a next generation 'semantic' Web [29]. As a first step, it was decided that the adoption of a common ontology language would facilitate semantic interoperability across the various projects making up the program. RDFS (which had already been proposed as a W3C standard) was seen as a good starting point, but was not sufficiently expressive to meet DAML's requirements. A new language called DAML-ONT was therefore developed that extended RDF with language constructors from object-oriented and frame-based knowledge representation languages.

DAML-ONT was tightly integrated with RDFS, and while this was useful from a compatibility viewpoint, it led to some serious problems in the design of the language. Like RDFS, DAML-ONT suffered from an inadequate semantic specification, and it was soon realised that this could lead to disagreements, both amongst humans and machines, as to the precise meaning of terms in a DAML-ONT ontology. Moreover,

DAML-ONT property restrictions had, like those of RDFS, global rather than local scope, and while this was reasonable for the domain and range constraints provided by RDFS, global cardinality constrains, for example, are difficult to understand and of doubtful utility; in fact it seems likely that this would have been recognised as a design flaw if the semantics of the language had been adequately formalised.

4.3. OIL

At around the same time that DAML-ONT was being developed, a group of (largely European) researchers with aims similar to those of the DAML researchers had designed another Web-oriented ontology language called OIL[4] (the Ontology Inference Layer) [3]. OIL was the first ontology language to combine elements from Description Logics, frame languages and Web standards such as XML and RDF. OIL placed a strong emphasis on formal rigor, and the language was explicitly designed so that its semantics could be specified via a mapping to the $\mathcal{SHIQ}$ Description Logic [19]. The structure of the language was, however, frame-based, using compound class 'definitions' in the style described in Section 3.2. OIL had both XML and RDF syntaxes, but although the RDF syntax was designed to maintain compatibility with RDFS, it did not concern itself with the precise details of RDF semantics, which had not at that time been formally defined.

4.4. DAML + OIL

It became obvious to both the DAML-ONT and OIL groups that their objectives could best be served by combining their efforts, the result being the merging of DAML-ONT and OIL to produce DAML + OIL. The development of DAML + OIL was undertaken by a committee largely made up of members of the two language design teams, and rather grandly titled the Joint US/EU ad hoc Agent Markup Language Committee.[5]

The merged language has a formal semantics given by its own DL style model theory instead of via a translation into a suitable DL. The DL-derived language constructors of OIL were retained in DAML + OIL,

---

[3] http://www.daml.org/.

[4] http://www.ontoknowledge.org/oil.

[5] http://www.daml.org/committee.

but the frame structure was largely discarded in favour of DL style axioms, which were more easily integrated with RDF syntax.

Influenced by DAML-ONT, DAML + OIL is more tightly integrated with RDF. DAML + OIL, however, only provided a meaning for those parts of RDF which were consistent with its own syntax and DL style model theory. This did not seem to be too much of a problem given that RDF did not at that time have a formally specified meaning of its own, but was the cause of serious difficulties when DAML + OIL was used as the basis for OWL.

5. Problems along the path

The multiple influences on OWL have lead to a number of problems. Some of these problems are a result of conflicting requirements, as in the conflict between using RDF/XML as the offcial OWL syntax and having an easy-to-read syntax. Some of these problems arise from a need to extend previous solutions, as in the problems arising in crafting an extension to RDF that incorporates information that does not fit well into the RDF world view.

5.1. Syntactic problems

For a number of reasons, including maintaining connections to frames and Description Logics, OWL should have an easy-to-read syntax that can be easily understood and easily created. However, it was a requirement of OWL that it use XML as its normative syntax, and, moreover, use XML in the same way as it is used in RDF [30]. This requirement had already been addressed by OIL and, later, by DAML + OIL: OIL has both an RDF/XML and an XML syntax [31], while DAML + OIL has only an RDF/XML syntax [4].

Taken just as a syntax for OWL, RDF in the form of RDF/XML has a number of problems. These problems can be overcome, but they do make OWL more complex that it might otherwise be.

One problem is that RDF/XML is extremely verbose. Compare for example, information about a class as it would be given in a Description Logic syntax

$$\text{Student} = \text{Person} \sqcap \; \geqslant 1 \text{enrolledIn}$$

(a `Student` is a `Person` who is `enrolledIn` at least one thing), with how it would most naturally be written using the OWL RDF/XML syntax[6]

```
<owl:Class rdf:ID=''Student''>
 <owl:intersectionOf rdf:parsetype
   =''Collection''>
  <owl:Class rdfs:about=''
    Person''/>
  <owl:Restriction>
   <owl:onProperty rdf:resource
     =''enrolledIn''/>
   <owl:minCardinality rdfs:
     datatype=''&xsd;Integer''>
     1
   </owl:minCardinality>
  </owl:Restriction>
 <owl:intersectionOf>
</owl:Class>
```

Another problem is that RDF breaks everything down into RDF triples [12].[7] This means that many OWL constructs, such as property restrictions, have to be encoded as several triples. OWL generally uses an encoding similar to that used by DAML + OIL. For example, an OWL value restriction that would be written in Description Logic syntax as $\exists \text{child.person}$ (the class whose instances have some `child` that is a `person`) is encoded as two RDF triples something like

```
_:x owl:onProperty ex:child.
_:x owl:someValuesFrom ex:Person.
```

where `_:x` is a syntactic placeholder for the restriction as a whole.

A third problem is that all RDF triples are independent. This means, for example, that as far as RDF is concerned there is no requirement that the two above triples must always occur together. Similarly, there is no requirement that there not be extra triples, so adding

---

[6] Full details on the OWL RDF/XML syntax can be found in the OWL reference document [1].

[7] The syntax for RDF triples here is one used in the RDF Semantics document [16]. An RDF triple is written as a subject, a predicate and an object, in that order. RDF URI references in triples are generally abbreviated as XML qualified names. RDF blank nodes (anonymous objects) are written with a leading '_:'.

```
_:x owl:onProperty ex:friend.
_:x owl:allValuesFrom ex:Doctor.
```

to the above two triples cannot be ruled out in RDF.

A fourth problem is that RDF triples are all accessible. This means that circular and other unusual structures cannot be ruled out. For example, there is no problem in RDF with collections of triples like

```
_:x owl:onProperty ex:child.
_:x owl:allValuesFrom _:x.
```

These issues are not addressed in OIL, which provides no guidance as to what should happen for collections of triples that do not match the syntax productions of the language. DAML + OIL takes a different approach, allowing unusual constructions but declining to give them a DAML + OIL meaning. OWL has roughly followed the DAML + OIL solution, but with several modifications.

5.2. Semantic problems

Once issues of syntax have been addressed, issues related to meaning still remain. RDF provides a meaning for every triple, so if OWL is to be considered to be an extension of RDF, the meaning that OWL provides for triples needs to be an extension of this RDF meaning.

This was not as much of an issue when OIL and DAML + OIL were designed, as the meaning of RDF was not very well specified. OIL in particular does not bother to relate the RDF meaning of its RDF/XML syntax to the OIL meaning of this syntax—the RDF/XML syntax for some OIL constructs does more-or-less line up with the RDF meaning of these constructs but this is by no means the case for all such constructs. For example, OIL has a special property (`oil:hasSlotConstraint`) used to relate a class to its slots, but the RDF meaning [16] of this property, namely the standard meaning assigned to any RDF triple is ignored by the OIL semantics.

DAML + OIL does a better job of abiding by the RDF meaning of its syntax. The DAML + OIL model theory [32] includes a semantic condition for triples that is close to the RDF meaning (as defined at that time) for triples. Further, DAML + OIL uses the built-in RDF and RDFS vocabulary to a greater extent than does OIL, and uses it in a way generally compati-

ble with the RDF or RDFS meaning (as defined at that time) for this vocabulary. For example DAML + OIL uses `rdfs:subClassOf` to relate classes to super-classes, including DAML + OIL restrictions, whereas OIL uses `oil:hasSlotConstraint` in some of these situations.

Even when DAML + OIL was being developed, however, there were some aspects of the meaning of RDFS that could not be reconciled with the appropriate meaning in DAML + OIL. In particular, RDFS [33] then had an unusual meaning for domains and ranges of properties. Only a single range was permitted for properties and multiple domains were treated disjunctively. For example,

```
ex:foo rdfs:domain ex:Person.
ex:foo rdfs:domain ex:Rock.
```

would allow both people and rocks to participate in the foo property.

This disjunctive reading of domains caused problems for the DAML + OIL semantics, so a choice was made to change this to a conjunctive reading and petition the newly-formed RDF core working group to change RDFS to allow multiple domains and ranges, both with a conjunctive reading. As part of its clean up of the RDF and RDFS semantics, the RDF core working group has decided to make this change, eliminating a problem for OWL.

While cleaning up problems with RDF and RDFS, the RDF core working group also decided to put RDF on a firmer semantic ground. It did this by providing a model theory for RDF and RDFS, along with a standard treatment of inference for RDF and RDFS. This has meant that there is now more meaning provided for RDF and RDFS that OWL has to be compatible with. In particular, all the triples that are used to encode the OWL syntax now have RDF meaning, and this RDF meaning has to be taken into account by OWL if the semantics of OWL are to be fully compatible with those of RDF and RDFS.

Neither OIL nor DAML + OIL provided a standard theory of inference. This was common in the formalisms that influenced OIL and DAML + OIL. Frames generally provided an interface to the internal data structures in lieu of any other inferences or even queries. Description Logics do provide a formal theory of querying, but this is somewhat different from a standard theory of inference.

The difference is that Description Logic querying could have been defined for DAML+OIL in a way that would have helped to hide the RDF meaning of triples. For example, asking whether an individual belonged to a class could add the syntax used to specify the class to the premises of the query. However, a standard theory of inferencing cannot do this.

The effects of this change can be seen in a simple example. Given the following information

```
ex:John rdf:type ex:Student.
ex:John rdf:type ex:Employee.
```

it would have been fairly easy to arrange it so that asking whether John belonged to the intersection of student and employee first ensured that this intersection existed and then asked whether John belonged to it. However, turning this into an entailment requires the above information to entail

```
_:c owl:intersectionOf _:l1.
_:l1 rdf:first ex:Student.
_:l1 rdf:rest _:l2.
_:l2 rdf:first ex:Employee.
_:l2 rdf:rest rdf:nil.
ex:John rdf:type _:c.
```

which, because of the RDF meaning ascribed to all triples, requires the existence of the triples that encode the syntax.[8]

OWL thus has had to develop a method that augments the new RDF semantics just enough to support the above inferences without being too strong, and thus ending up with semantic paradoxes (this will be discussed in more detail in Section 6.4).

5.3. Expressive power

Because many things were expected of OWL (see the long list of design goals, requirements and objectives in [5]), there were many demands for expressive power going beyond that generally provided by Description Logics. For example, many users wanted to be able to associate information with classes and properties and to make classes belong to other classes, as is possible in RDF. Similarly, there were many demands for expressive power going beyond RDF and RDFS. For example, many users wanted to be able to provide local typing for property values, as is possible in Description Logics. OWL had to be designed to somehow allow these sorts of expressivity while still retaining connections to its roots.

When DAML + OIL was developed, the only datatype supported by RDF was literals: roughly undifferentiated values given as strings. DAML + OIL thus had to provide its own solution for datatypes, and did so by allowing the use of XML schema datatypes [34]. However, any reasonable solution to datatyping that uses only RDF syntax needs help from RDF (i.e. an extension to RDF syntax), and thus the DAML + OIL solution remained incomplete.

Recently RDF has added its own version of datatyping, similar to, but different from, the DAML+OIL solution. OWL has thus needed to move from DAML + OIL datatyping to RDF datatyping.

5.4. Computational problems

One aspect of OWL that distinguishes it from RDF and RDFS is that it supports a rich set of inferences. Some of these inferences are quite obvious, such as the example given above about students and employees, and thus appear to be easy to compute. Other inferences supported by OWL, however, are quite complex, requiring, e.g. reasoning by cases and following chains of properties.

Taking all the representational desires for OWL together would have created a formalism where key inference problems were undecidable. For example, allowing relationships to be asserted between property chains (such as saying that an uncle is precisely a parent's brother) would make OWL entailment undecidable.[9] In addition, some aspects of RDF, such as the use of classes as instances, interact with the rest of OWL to create computational difficulties, taking OWL beyond the range of practical algorithms and implemented reasoning systems.

---

[8] Note that OWL constructs such as `intersectionOf` and `unionOf` are encoded using RDF lists constructed of `rdf:first` and `rdf:rest` triples; see the OWL reference document [1] for full details.

[9] It is easy to show that, if extended in this way, OWL could be used to encode the word problem, which is well known to be undecidable [38].

OWL has thus had to provide a solution for these computational issues while still retaining upwards compatibility with RDF and RDFS.

6. Solutions

The Web Ontology Working Group spent the better part of a year overcoming the basic tensions underlying the above problems. The difficulty lay not in each problem in isolation, but in the combination of all the above problems and the constraints placed on the design of OWL. It would have been much easier, for example, to meet all the above requirements if only

```
Class(Student  complete
               Person
               restriction(enrolledIn minCardinality(1))).
```

OWL could have used an extension of the RDF syntax. If this had been allowed, OWL could have added new, natural syntax for its constructs whose semantics would not have been required to carry along an RDF triple meaning.

Nevertheless a viable solution has been found that satisfies all the above requirements. Or, actually, it is more accurate to say that three solutions have been found, each of which satisfies almost all of the above requirements.

1. *OWL DL*: If friendly syntax or decidable inference is considered of primary importance, then OWL DL, a version of OWL with decidable inference that can be written in a frame or Description Logic manner, is appropriate.
2. *OWL Lite*: If an even simpler syntax and more tractable inference is considered of primary importance, then OWL Lite, a syntactic subset of OWL DL, is appropriate.
3. *OWL Full*: If compatibility with RDF and RDFS is considered of primary importance, then OWL Full, a syntactic and semantic extension of RDFS, is appropriate.

The next section provides a more-detailed description of these versions (species) of OWL, and explains how the problems described above have been overcome.

6.1. Readability

As shown by the examples above, OWL is not very readable when written as RDF/XML or even as RDF triples. Part of this problem is that RDF/XML is extremely verbose, but the major part of the readability problem is the encoding of OWL constructs into RDF/XML or RDF triples.

In part to address this problem, an abstract syntax (cf., Section 7.1) was created for OWL, along with a mapping from abstract syntax to RDF graphs. This abstract syntax is closer to the syntax of OIL than of DAML + OIL, but without OIL's extreme emphasis on readability. In this abstract syntax, the Student example above would be written

OWL DL was then defined as the syntactic subset of OWL induced by the translation from the abstract syntax to RDF graphs. That is, an RDF graph is an OWL DL ontology just when it is the translation of some ontology in the abstract syntax. Users and tools that are more interested in readability than in RDF/XML can use this abstract syntax internally, or even externally for presentation to users, reserving the RDF/XML syntax for purposes of exchange between applications.

6.2. Handling malformed graphs

Because OWL Full allows arbitrary RDF graphs, it must be able to handle malformed OWL syntax. (OWL DL does not suffer from this problem as it is defined in terms of the necessarily well-formed RDF graphs that can be generated from the abstract syntax.) OWL uses an extension of the DAML + OIL solution: only triples that together form well-formed OWL constructs are given an extra meaning, so

```
_:x owl:onProperty ex:child.
```

by itself does not have any special OWL meaning.

To handle the cases of too many triples, OWL again uses the DAML + OIL solution of picking out all the well-formed subsets and giving them OWL meaning. This has unusual consequences, for example

```
_:x owl:onProperty ex:child.
_:x owl:someValuesFrom ex:Person.
_:x owl:onProperty ex:friend.
_:x owl:allValuesFrom ex:Doctor.
```

ends up equating the extension of four different OWL restrictions (all possible combinations of the two properties with the two classes), which is almost certainly not what was intended by the user. This solution, however, maintains monotonicity, and the (possibly) non-intuitive meaning is a minor problem given that such malformed constructions can easily be avoided.

The lack of structure in RDF graphs has to be handled by semantic means, which are described next.

6.3. Providing a viable semantic theory for OWL

As RDF now has a model theory, with a full-fledged notion of entailment, OWL has to provide an upward-compatible model theory that appropriately handles entailments over the OWL constructs. This would have been easy if OWL had been able to extend the RDF syntax, as then these new bits of syntax could have had an OWL-only meaning. However, it was a requirement that OWL had an RDF syntax, and that this syntax carried all of its RDF meaning. This two-way compatibility requirement is much stronger than that usually imposed between weaker formalisms (like propositional logic) and stronger formalisms (like first-order logic) where the stronger formalism is allowed to extend the syntax of the weaker formalism.

The most severe aspect of this problem is that OWL syntactic constructs that are encoded as multiple triples have to retain the RDF meaning for these triples. As all RDF triples carry semantic conditions, they cannot just be inferred from nothing. Instead, the OWL semantics has had to add special constraints that essentially state that every OWL interpretation must include certain constructs. (Such constraints are usually called comprehension principles.) For example, one comprehension principle for OWL states that every model must include all finite lists of classes; another states that every such list must have a corresponding intersection class, by requiring that there is some class that is connected to the list by an `owl:intersectionOf` property.

These comprehension principles support the entailment from

```
ex:John rdf:type ex:Student.
ex:John rdf:type ex:Employee.
```

to

```
_:c owl:intersectionOf _:l1.
_:l1 rdf:first ex:Student.
_:l1 rdf:rest _:l2.
_:l2 rdf:first ex:Employee.
_:l2 rdf:rest rdf:nil.
ex:John rdf:type _:c.
```

because `_:l1` can be the required list of `Student` and `Employee`, `_:l2` can be the required tail of this list, and `_:c` can be the required intersection of the list. Additional (and more-usual) semantic conditions require that `ex:John` belong to `_:c`, finishing all that is required for the entailment to hold.

6.4. Avoiding paradoxes

Comprehension principles are very powerful, as they create something from nothing (or, at least, something from very little). This power can easily lead to serious problems.

For example, the Russell paradox is a paradox precisely because of the comprehension principles built into an early version of set theory. This early version of set theory had a comprehension principle that stated that a set could be constructed of the things that satisfied a formula with one free variable, e.g. the formula of being a human, $x \in$ human, could be used to construct the set of humans, $\{x | x \in \text{human}\}$. Unfortunately, from the formula of not belonging to oneself, $x \notin x$, the set $\{x | x \notin x\}$ arises. This set causes problems wherever it exists because it is impossible to determine whether it belongs to itself. The comprehension principle mandates that it exists everywhere, thus causing this early version of set theory to collapse.

A similar situation can arise with OWL. It seems natural to want to have circular OWL-like constructs, for example classes whose instances are related only to other instances of the class, such as in

```
_:c owl:onProperty ex:child.
_:c owl:allValuesFrom _:c.
```

that might be used in a naive representation of some aspects of biology.

However, having comprehension principles for such circular classes can easily lead to a requirement for the existence of classes like

```
_:c owl:onProperty rdf:type.
_:c owl:allValuesFrom _:d.
_:d owl:complementOf _:l.
_:l rdf:first _:c.
_:l rdf:rest rdf:nil.
```

which is the class of things that have no type relationship to the class itself. Objects that belong to this class can not belong to it, and vice versa, so if the comprehension principles required the existence of this class, then every OWL ontology would be paradoxical.

To avoid these paradoxes, the OWL comprehension principles never require the existence of circular chains of reference like the one above. However, this does mean that there are entailments that one might expect, such as ones involving the construct with ex:child above, for example having a person with no children belong to such a construct, that are not valid in OWL. Devising these comprehension principles took a surprising amount of effort (much of which involved determining the ground rules for the principles).

6.5. Retaining decidability

OWL Full is undecidable (for a number of reasons), and even OWL DL could easily be undecidable if it included certain constructs known to cause undecidability in Description Logics (see, e.g. [19]). Therefore, OWL DL was carefully crafted to remain decidable, and does not include, for example, relationships between role chains, which would cause undecidability by embedding the word problem in OWL DL.

This is not to say that inference in OWL DL is not hard. OWL DL has a difficult entailment problem, as inference in $\mathcal{SHOIN}(\mathbf{D})$ is of worst-case nondeterministic exponential time (NExpTime) complexity [35], and OWL DL should have the same complexity. Worse, there is as of yet no known 'practical' complete algorithm for inference in $\mathcal{SHOIN}(\mathbf{D})$, i.e. one that is likely to perform well on the kinds of problem encountered in typical applications. In default of such an algorithm, the behaviour of OWL DL reasoners is likely to be less predictable (both in terms of the time taken to respond to queries, and the use of system re-

sources), and they may sometimes return the answer 'unknown' in response to queries.

OWL Lite is better in this regard. Inference in $\mathcal{SHIF}(\mathbf{D})$ is of worst-case deterministic exponential time (ExpTime) complexity [35], and OWL Lite has the same complexity. Moreover, there are practical optimized algorithms for inference in OWL Lite, such as the algorithm underlying the Description Logic systems FaCT [36] and RACER [37]. These systems have been shown to work well in realistic applications and to be able to reason with large ontologies.

7. OWL

This section describes how the solutions outlined above have been incorporated in the final design of the OWL language. It is not intended as a full description of the language, for this, readers should turn to the W3C documents on OWL [1,6,26].

For various reasons, described in the preceding sections, there are two styles of using OWL. In the first style, embodied in OWL DL and OWL Lite, only certain constructions are allowed, and these constructions can only be combined in certain ways. The benefits of staying within these limitations include decidability of inferences and the possibility of thinking of OWL in a more-standard fashion, essentially as an expressive Description Logic. In the second style, embodied in OWL Full, all RDF graphs are allowed. The benefits of this expansive style include total upward compatibility with RDF and a greater expressive power.

Even the more-limited versions of OWL have some differences from standard Description Logics. These differences move these versions of OWL from the formal Description Logic world to the Semantic Web world.

- OWL uses URI references as names, and constructs these URI references in the same manner as that used by RDF. It is thus common in OWL to use qualified names as shorthands for URI references, using, for example, the qualified name owl:Thing for the URI reference http://www.w3.org/2002/07/owl#Thing.
- OWL gathers information into ontologies, which are generally stored as Web documents written in RDF/XML. Ontologies can import other ontologies,

adding the information from the imported ontology to the current ontology.

- Even the DL/Lite style of using OWL allows RDF annotation properties to be used to attach information to classes, properties, and ontologies, such as `owl:DeprecatedClass`. These annotations are RDF triples, and are therefore required to carry a full semantic weight. They cannot be treated as informal comments without a formal meaning. This partly breaks down the firm Description Logic distinction between individuals, on the one hand, and classes and properties, on the other.
- OWL uses the facilities of RDF datatypes and XML schema datatypes to provide datatypes and data values.

- The DL and Lite versions of OWL have a frame-like abstract syntax, whereas RDF/XML is the official exchange syntax for all of OWL.

7.1. OWL as a Description Logic

OWL DL—the Description Logic style of using OWL—is very close to the $\mathcal{SHOIN}(\mathbf{D})$ Description Logic which is itself an extension of the influential $\mathcal{SHOQ}(\mathbf{D})$ Description Logic [20] (extended with inverse roles and restricted to unqualified number restrictions). OWL DL can form descriptions of classes, datatypes, individuals and data values using the constructs shown in Fig. 1. In this table the first column gives the OWL abstract syntax for the construction,

Abstract Syntax	DL Syntax	Semantics
Descriptions (C)		
A (URI reference)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
`owl:Thing`	$\top$	$\texttt{owl:Thing}^{\mathcal{I}} = \Delta^{\mathcal{I}}$
`owl:Nothing`	$\bot$	$\texttt{owl:Nothing}^{\mathcal{I}} = \{\}$
`intersectionOf`(C_1 C_2 ...)	$C_1 \sqcap C_2$	$(C_1 \sqcap D_1)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap D_2^{\mathcal{I}}$
`unionOf`(C_1 C_2 ...)	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
`complementOf`(C)	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
`oneOf`(o_1 ...)	$\{o_1, \ldots\}$	$\{o_1, \ldots\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots\}$
`restriction`(R `someValuesFrom`(C))	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
`restriction`(R `allValuesFrom`(C))	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \to y \in C^{\mathcal{I}}\}$
`restriction`(R `hasValue`(o))	$R : o$	$(\forall R.o)^{\mathcal{I}} = \{x \mid \langle x, o^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$
`restriction`(R `minCardinality`(n))	$\geqslant n\,R$	$(\geqslant n\,R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) \geqslant n\}$
`restriction`(R `minCardinality`(n))	$\leqslant n\,R$	$(\geqslant n\,R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) \leqslant n\}$
`restriction`(U `someValuesFrom`(D))	$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in U^{\mathcal{I}} \text{ and } y \in D^{\mathbf{D}}\}$
`restriction`(U `allValuesFrom`(D))	$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in U^{\mathcal{I}} \to y \in D^{\mathbf{D}}\}$
`restriction`(U `hasValue`(v))	$U : v$	$(U : v)^{\mathcal{I}} = \{x \mid \langle x, v^{\mathcal{I}}\rangle \in U^{\mathcal{I}}\}$
`restriction`(U `minCardinality`(n))	$\geqslant n\,U$	$(\geqslant n\,U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) \geqslant n\}$
`restriction`(U `maxCardinality`(n))	$\leqslant n\,U$	$(\leqslant n\,U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) \leqslant n\}$
Data Ranges (D)		
D (URI reference)	D	$D^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}^{\mathcal{I}}$
`oneOf`(v_1 ...)	$\{v_1, \ldots\}$	$\{v_1, \ldots\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, \ldots\}$
Object Properties (R)		
R (URI reference)	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	R^-	$(R^-)^{\mathcal{I}} = (R^{\mathcal{I}})^-$
Datatype Properties (U)		
U (URI reference)	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}^{\mathcal{I}}$
Individuals (o)		
o (URI reference)	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Data Values (v)		
v (RDF literal)	v	$v^{\mathcal{I}} = v^{\mathbf{D}}$

Fig. 1. OWL DL descriptions, data ranges, properties, individuals and data values.

Abstract Syntax	DL Syntax	Semantics
`Class(A partial `C_1` ...`C_n`)`	$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
`Class(A complete `C_1` ...`C_n`)`	$A = C_1 \sqcap \ldots \sqcap C_n$	$A^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
`EnumeratedClass(A `o_1` ...`o_n`)`	$A = \{o_1, \ldots, o_n\}$	$A^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots, o_n^{\mathcal{I}}\}$
`SubClassOf(`C_1` `C_2`)`	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
`EquivalentClasses(`C_1` ...`C_n`)`	$C_1 = \ldots = C_n$	$C_1^{\mathcal{I}} = \ldots = C_n^{\mathcal{I}}$
`DisjointClasses(`C_1` ...`C_n`)`	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} \{\}, i \neq j$
`Datatype(`D`)`		$D^{\mathcal{I}} \subseteq \Delta_{\mathbf{D}}^{\mathcal{I}}$
`DatatypeProperty(U super(`U_1`)...super(`U_n`)`	$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$
`    domain(`C_1`) ...domain(`C_m`)`	$\geqslant 1\, U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathbf{D}}^{\mathcal{I}}$
`    range(`D_1`) ...range(`D_l`)`	$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$
`    [Functional])`	$\top \sqsubseteq\, \leqslant 1\, U$	$U^{\mathcal{I}}$ is functional
`SubPropertyOf(`U_1` `U_2`)`	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
`EquivalentProperties(`U_1` ...`U_n`)`	$U_1 = \ldots = U_n$	$U_1^{\mathcal{I}} = \ldots = U_n^{\mathcal{I}}$
`ObjectProperty(R super(`R_1`)...super(`R_n`))`	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$
`    domain(`C_1`) ...domain(`C_m`)`	$\geqslant 1\, R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
`    range(`C_1`) ...range(`C_l`)`	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$
`    [inverseOf(`R_0`]`	$R = (^-R_0)$	$R^{\mathcal{I}} = (R_0^{\mathcal{I}})^-$
`    [Symmetric]`	$R = (^-R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^-$
`    [Functional]`	$\top \sqsubseteq\, \leqslant 1\, R$	$R^{\mathcal{I}}$ is functional
`    [InverseFunctional]`	$\top \sqsubseteq\, \leqslant 1\, R^-$	$(R^{\mathcal{I}})^-$ is functional
`    [Transitive])`	$Tr(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$
`SubPropertyOf(`R_1` `R_2`)`	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
`EquivalentProperties(`R_1` ...`R_n`)`	$R_1 = \ldots = R_n$	$R_1^{\mathcal{I}} = \ldots = R_n^{\mathcal{I}}$
`AnnotationProperty(`S`)`		
`Individual(o type(`C_1`) ...type(`C_n`)`	$o \in C_i$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$
`    value(`R_1` `o_1`)...value(`R_n` `o_n`)`	$\langle o, o_i \rangle \in R_i$	$\langle o^{\mathcal{I}}, o_i^{\mathcal{I}} \rangle \in R_i^{\mathcal{I}}$
`    value(`U_1` `v_1`)...value(`U_n` `v_n`))`	$\langle o, v_i \rangle \in U_i$	$\langle o^{\mathcal{I}}, v_i^{\mathcal{I}} \rangle \in U_i^{\mathcal{I}}$
`SameIndividual(`o_1` ...`o_n`)`	$o_1 = \ldots = o_n$	$o_i^{\mathcal{I}} = o_j^{\mathcal{I}}$
`DifferentIndividuals(`o_1` ...`o_n`)`	$o_i \neq o_j, i \neq j$	$o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}, i \neq j$

Fig. 2. OWL DL axioms and facts.

while the second column gives the standard Description Logic syntax.

OWL DL uses these description-forming constructs in axioms that provide information about classes, properties and individuals, as shown in Fig. 2. Again, the frame-like abstract syntax is given in the first column, and the standard Description Logic syntax is given in the second column.

7.2. Semantics for OWL DL

A formal semantics, very similar to the semantics provided for Description Logics (see Section 3.1), is provided for this style of using OWL. Full details on this model theory can be found in the OWL Semantics and Abstract Syntax [26].

Because OWL includes datatypes, the semantics for OWL is very similar to that of Description Logics that also incorporate datatypes, in particular $\mathcal{SHOQ}(\mathbf{D})$. However, the particular datatypes used in OWL are taken from RDF and XML schema datatypes [34]. Data values such as "44"$^{\wedge\wedge}$`xsd:integer` thus mean what they would mean as XML schema data values.

The specific meaning given to OWL DL descriptions is shown in the third column of Fig. 1, where $\Delta^{\mathcal{I}}$ is the domain of individuals in a model and $\Delta_{\mathbf{D}}^{\mathcal{I}}$ is the domain of data values. As usual, the meaning of axioms is given in terms of constraints on models, as shown in the third column of Fig. 2.

The semantics for OWL DL does include some unusual (for Description Logics) aspects. Annotations

are given a simple separate meaning, not shown here, that can be used to associate information with classes, properties and individuals in a manner compatible with the RDF semantics. Ontologies also live within the semantics and can be given annotation information. Finally, `owl:imports` is given a meaning that involves finding the referenced ontology (if possible) and adding its meaning to the meaning of the current ontology.

What makes OWL DL a Semantic Web language, therefore, is not its semantics, which are quite standard for a Description Logic, but instead the use of URI references for names, the use of XML schema datatypes for data values, and the ability to connect to documents in the World Wide Web.

7.3. An easier OWL: OWL Lite

OWL DL is related to $\mathcal{SHOIN}(\mathbf{D})$, a very expressive Description Logic. This Description Logic is somewhat difficult to present to naive users, as it is possible to build complex boolean descriptions using, for example, union and complement. $\mathcal{SHOIN}(\mathbf{D})$ is also difficult to reason with, as key inference problems have NExpTime complexity, and somewhat difficult to build even non-reasoning tools for, because of the complex descriptions.

For these reasons, a subset of OWL DL has been identified that should be easier on all the above metrics; this subset is called OWL Lite. OWL Lite prohibits unions and complements, restricts intersections to the implicit intersections in the frame-like class axioms, limits all embedded descriptions to concept names, does not allow individuals to show up in descriptions or class axioms, and limits cardinalities to 0 or 1.

These restrictions make OWL Lite similar to the Description Logic $\mathcal{SHIF}(\mathbf{D})$. Like $\mathcal{SHIF}(\mathbf{D})$, key inferences in OWL Lite can be computed in worst case exponential time (ExpTime), and there are already several optimized reasoners for logics equivalent to OWL Lite (see Section 6.5). This improvement in tractability comes with relatively little loss in expressive power. Although OWL Lite syntax is more restricted than that of OWL DL, it is still possible to express complex descriptions by introducing new class names and exploiting the implicit negations introduced by disjointness axioms. Using these techniques, all OWL DL descrip-

tions can be captured in OWL Lite except those containing either individual names or cardinalities greater than 1.

7.4. OWL Full as an RDF extension

OWL DL and OWL Lite are extensions of a restricted use of RDF and RDFS, because, unlike RDF and RDFS, they do not allow classes to be used as individuals, and the language constructors cannot be applied to the language itself. For users who need these capabilities, a version of OWL that is upward compatible with RDF and RDFS has been provided; this version is called OWL Full. In OWL Full, all RDF and RDFS combinations are allowed. For example, in OWL Full, it is possible to impose a cardinality constraint on `rdfs:subClassOf`, if so desired.

OWL Full contains OWL DL, but goes well outside the standard Description Logic framework. The penalty to be paid here is twofold. First, reasoning in OWL Full is undecidable (because restrictions required in order to maintain the decidability of OWL DL do not apply to OWL full [19]). Second, the abstract syntax for OWL DL is inadequate for OWL Full, and the official OWL exchange syntax, RDF/XML, must be used.

7.5. Semantics for OWL Full

OWL Full has been given a model-theoretic semantics that is a vocabulary extension of the RDF model theory [16,26]. A correspondence between this semantics and the semantics of OWL DL has also been established: it has been shown that the model theory for OWL DL has the same consequences as this RDF-style model theory for those OWL ontologies that can be written in the OWL DL abstract syntax [26].

The correspondence means that, given two OWL DL ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, written in the abstract syntax, $\mathcal{O}_1$ entails $\mathcal{O}_2$ according to the OWL DL model theory if and only if the mapping of $\mathcal{O}_1$ into RDF triples entails the mapping of $\mathcal{O}_2$ into RDF triples according to the OWL Full model theory. The proof of this correspondence is rather complex, and could break down, e.g. as a result of (apparently) minor changes in the specification of OWL or RDF. In view of the relative fragility of this correspondence, and in order to avoid any possible confusion as to the meaning of

OWL DL, the OWL Full model theory has been given 'non-normative' status (i.e. it is only informative) for OWL ontologies that can be written in the abstract syntax. This means that the OWL DL model theory will be taken as definitive should the correspondence break down or be shown to be incomplete.

8. Future extensions

Clearly, OWL is not the final word on ontology languages for the Semantic Web. A number of features were already identified in the OWL requirements document [5], and many others are under discussion. In this section, we briefly (and non-exhaustively) list a few of these possible extensions and improvements to OWL as it stands.

- *Modules and imports*. Importing ontologies defined by others will be the norm on the Semantic Web. However, the importing facility of OWL is very trivial: it only allows to import an entire ontology, specified by location. Even if one would only want to use a small portion of another ontology, one is forced to import that entire ontology. Module constructions in programming languages are based on a notion of information hiding: the module promises to provide some functionality to the outside world (the export-clause of the module), but the importing module need not concern itself with how this functionality is achieved. It is an open research question what a corresponding notion of information hiding for ontologies could be, and how it could be used as the basis for a good import construction.
- *Defaults*. Many practical knowledge representation systems allow inherited values to be overridden by more specific classes in the hierarchy, treating the inherited values as defaults. Although widely used in practical knowledge representation systems, no consensus has been reached on the right formalisation for the non-monotonic behaviour of default values.
- *Closed world assumption*. The semantic of OWL currently adopts the standard logical model of an open world assumption: a statement cannot be assumed true on the basis of a failure to prove it. Clearly, on the huge and only partially knowably World Wide Web this is the correct assumption.

Nevertheless, the opposite approach (a closed world assumption: a statement is true when its negation cannot be proven) is also useful in certain applications. The closed world assumption is closely tied to the notion of defaults, and leads to the same non-monotonic behaviour, reason for it not to be included in OWL as it stands.
- *Unique names assumption*. Typical database applications assume that individuals with different names are indeed different individuals. OWL follows the usual logical paradigm where this is not the case. If two individuals (or classes, or properties) have different names, we may still derive by inference that they must be the same. As with the non-closed world assumption, the non-unique names assumption is the most plausible one to make on the World Wide Web, but as before, situations exist where the unique names assumption is useful. More subtly, one may want to indicate portions of the ontology for which the assumption does or does not hold.
- *Procedural attachment*. A common concept in knowledge representation is to define the meaning of a term not through explicit definitions in the language (as is done in OWL), but by attaching a piece of code to be executed for computing the meaning of the term. Although widely used, this concept does not lend itself very well to integration in a system with a formal semantics, and has not been included in OWL.
- *Property chaining, rules*. As explained above, for reasons of decidability, OWL currently does not allow the composition of properties, but of course in many applications this is a useful operation. Even more general, one would want to define properties as general rules (Horn or otherwise) over other properties. Such integration of rule-based knowledge representation and DL-style knowledge representation is currently an active area of research.

9. Summary

Because of the ambitious design goals for OWL, because of the multiple influences on OWL, and also because of the structural requirements constraining OWL, the development of OWL has not been without problems. Through hard work and compromise, these problems have largely been overcome, resulting in a

ontology language that is truly part of the Semantic Web.

It was not possible to simultaneously satisfy all of the constraints on OWL, so two styles of using OWL have been developed, each suitable under different circumstances.

If an expressive ontology language with decidable inference is the main concern, then the OWL DL style is indicated. This style of using OWL loses some compatibility with RDF, mostly having to do with using classes and properties as individuals, but retains an expressive and useful ontology language. OWL DL also has a frame-like alternative syntax that can be used to make working with OWL easier.

Even though OWL DL is close to Description Logics, it includes features that firmly place it in the Semantic Web. OWL DL uses the datatyping mechanisms from RDF and many of the built-in XML schema datatypes. OWL DL uses RDF URI references as names, including the names from RDF, RDFS and XML schema datatypes that are relevant. Entailment in OWL DL is compatible with entailment in RDF and RDFS.

If a simpler ontology language is the main concern, then the OWL Lite subset of OWL DL can be used. This sub-language eliminates some of the things that can be said in OWL DL, but has effectively tractable inference, closely related to the inference already implemented in several Description Logic systems, such as FaCT [36] and RACER [37].

If, on the other hand, upward compatibility with RDF is the main concern, then the OWL Full style is indicated. This style extends RDF and RDFS to a full ontology language, with a well-specified entailment relationship that extends entailment in RDF and RDFS, while avoiding any paradoxes that might arise. However, entailment in OWL Full is undecidable, which can be a significant issue in some circumstances. Also, the user-friendly alternative syntax is not adequate for OWL Full, so RDF/XML must be used for OWL Full.

These styles of using OWL provide an ontology layer for the Semantic Web, significantly extending the capabilities of RDF and RDFS, and expanding the usefulness of the Semantic Web.

There remain, of course, significant issues that are deliberately not handled by OWL, but which are definitely relevant to many Semantic Web use cases:

- OWL avoids anything related to nonmonotonicity (such as default reasoning and localised closed world assumptions);
- OWL's limited expressiveness excludes operations such as property-chaining, or, more generally, axioms with variables (such as rules);
- OWL's import mechanism is rather crude, and does not support fine-grained operations (such as the importation of parts of ontologies); and
- OWL integrates datatypes in a very clean way, but there is no notion of operations on these datatypes (such as integer arithmetic or string operations).

Extending the current Semantic Web with some or all of these features will require not only a standardisation effort, but sets a significant research challenge to the community.

Acknowledgements

References

[1] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, OWL Web Ontology Language Reference, W3C Working Draft, 31 March 2003. Available at http://www.w3.org/TR/2003/WD-owl-ref-20030331.

[2] J. Heflin, J. Hendler, S. Luke, SHOE: A Knowledge Representation Language for Internet Applications, Technical Report CS-TR-4078, Department of Computer Science, University of Maryland, 1999.

[3] D. Fensel, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, OIL: an ontology infrastructure for the Semantic Web, IEEE Intell. Syst. 16 (2) (2001) 38–45.

---

[4] D. Connolly, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, DAML + OIL (March 2001) Reference Description, W3C Note, 18 December 2001. Available at http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218.

[5] J. Heflin, Web Ontology Language (OWL) Use Cases and Requirements, W3C Working Draft, 31 March 2003. Available at http://www.w3.org/TR/2003/WD-webontreq-20030331/.

[6] M.K. Smith, C. Welty, D. McGuinness, OWL Web Ontology Language Guide, W3C Working Draft, 31 March 2003. Available at http://www.w3.org/TR/2003/WD-owl-guide-20030331/.

[7] T. Berners-Lee, Weaving the Web, Harper, San Francisco, 1999.

[8] D.L. McGuinness, Ontological issues for knowledge-enhanced search, in: Proceedings of FOIS, Frontiers in Artificial Intelligence and Applications, IOS-press, Amsterdam, 1998.

[9] D.L. McGuinness, Ontologies for electronic commerce, in: Proceedings of the Artificial Intelligence for Electronic Commerce Workshop (AAAI'99), Orlando, FL, USA, July 18–22, 1999.

[10] S. McIlraith, T.C. Son, H. Zeng, Semantic Web services, IEEE Intell. Syst. 16 (2) (2001) 46–53.

[11] L. Li, I. Horrocks, A Software Framework for Matchmaking Based on Semantic Web Technology, in: Proceedings of the Twelfth International World Wide Web Conference (WWW'2003), Budapest, Hungary, May 20–24, 2003, pp. 331–339. ACM, 2003.

[12] G. Klyne, J.J. Carroll, Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Working Draft, 2003. Available at http://www.w3.org/TR/2003/WD-rdf-concepts-20030123.

[13] D. Brickley, R.V. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft, 23 January 2003. Available at http://www.w3.org/TR/2003/WD-rdf-schema-20030123/.

[14] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.), The Description Logic Handbook, Cambridge University Press, Cambridge, London, 2002.

[15] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, I. Horrocks, Enabling knowledge representation on the Web by extending RDF schema, in: Proceedings of the 10th World Wide Web Conference (WWW'10), Honolulu, Hawai, USA, May 2001, pp. 467–478.

[16] P. Hayes, RDF Semantics, W3C Working Draft, 2003. Available at http://www.w3.org/TR/2003/WD-rdf-mt-20030123.

[17] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for very expressive Description Logics, J. Interest Group Pure Appl. Logic 8 (3) (2000) 239–264.

[18] M. Schmidt-Schauß, G. Smolka, Attributive concept descriptions with complements, Artif. Intell. 48 (1) (1991) 1–26.

[19] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for expressive Description Logics, in: H. Ganzinger, D. McAllester, A. Voronkov (Eds.), Proceedings of the Sixth International Conference on Logic for Programming and Automated Reasoning (LPAR'99), Lecture Notes in Artificial Intelligence, vol. 1705, Springer, Berlin, 1999, pp. 161–180.

[20] I. Horrocks, U. Sattler, Ontology reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'2001), Seattle, Washington, USA, August 4–10, 2001, pp. 199–204.

[21] C. Lutz, The Complexity of Reasoning with Concrete Domains, Ph.D. Thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2002.

[22] W.E. Grosso, H. Eriksson, R.W. Fergerson, J.H. Gennari, S.W. Tu, M.A. Musen, Knowledge modelling at the millenium (The design and evolution of Protégé-2000), in: Proceedings of the Knowledge Acqusition Workshop (KAW'99), Banff, Alberta, Canada, October 16–21, 1999.

[23] V.K. Chaudhri, A. Farquhar, R. Fikes, P.D. Karp, J. Rice, OKBC: A programmatic foundation for knowledge base interoperability, in: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98), Madison, Wisconsin, USA, July 26–30, 1998, pp. 600–607.

[24] P.D. Karp, V.K. Chaudhri, J. Thomere, XOL: An XML-Based Ontology Exchange Language, Technical Report SRI AI Technical Note 559, SRI International, Menlo Park, CA, USA, 1999.

[25] S. Decker, D. Fensel, F. van Harmelen, I. Horrocks, S. Melnik, M. Klein, J. Broekstra, Knowledge representation on the Web, in: Proceedings of the 2000 Description Logic Workshop (DL'2000), Aachen, Germany, August 17–19, 2000, pp. 89–97. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-33/.

[26] P.F. Patel-Schneider, P. Hayes, I. Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax, W3C Working Draft, 31 March 2003. Available at http://www.w3.org/TR/2003/WD-owl-semantics-20030331/.

[27] F. Manola, E. Miller, Rdf Primer, W3C Working Draft, 23 January 2003. Available at http://www.w3.org/TR/rdf-primer/.

[28] J. Heflin, J. Hendler, Dynamic ontologies on the Web, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'2000), AAAI/MIT Press, Menlo Park, CA, 2000, pp. 443–449.

[29] J. Hendler, D.L. McGuinness, The DARPA Agent Markup Language, IEEE Intell. Syst. 15 (6) (2000) 67–73.

[30] D. Beckett, RDF/XML Syntax Specification (Revised), W3C Working Draft, 2002. Available at http://www.w3.org/TR/2002/WD-rdf-syntax-grammar-20021108.

[31] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, E. Motta, OIL: The Ontology Inference Layer, Technical Report IR-479, Faculty of Sciences, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, September 2000.

[32] F. van Harmelen, I. Horrocks, P.F. Patel-Schneider, A Model-theoretic Semantics for DAML + OIL (March 2001), W3C Note, 18 December 2001. Available at http://www.w3.org/TR/2001/NOTE-daml+oil-model-20011218.

[33] D. Brickley, R.V. Guha, Resource Description Framework (RDF) Schema Specification 1.0, W3C Recommendation, 27

March 2000. Available at http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[34] P.V. Biron, A. Malhotra, XML Schema Part 2: Datatypes, W3C Recommendation, 2001. Available at http://www.w3.org/TR/2002/WD-xmlschema-2-20010502/.

[35] S. Tobies, Complexity Results and Practical Algorithms for Logics in Knowledge Representation, Ph.D. Thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2001.

[36] I. Horrocks, Using an expressive Description Logic: FaCT or fiction? in: Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2–5, 1998, pp. 636–647.

[37] V. Haarslev, R. Möller, RACER system description, in: Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001), Lecture Notes in Artificial Intelligence, vol. 2083, Springer, Berlin, 2001, pp. 701–705.

[38] M. Wessel, Obstacles on the way to qualitative spatial reasoning with Description Logics: some undecidability results, in: Proceedings of the 2001 Description Logic Workshop (DL'2001), Stanford, California, USA, August 1–3, 2001.

Biographies

Ian Horrocks received his PhD from the University of Manchester in 1997, and is now a professor in computer science at the same institution. His primary research interest is knowledge representation, in particular ontologies and ontology languages, tableaux algorithms for Description Logics, optimisation techniques for such algorithms and the application of all of the above to the Semantic Web. He designed and implemented the well-known FaCT reasoner, and has been involved in the development of the OIL, DAML + OIL and OWL ontology languages. Ian has given numerous invited talks, as well as keynote presentations at several international conferences, including CADE and EDBT. He has published widely (see http://www.cs.man.ac.uk/~horrocks/), and is on the programme committees and editorial boards of numerous conferences and journals; he was the program chair of the 2002 International Semantic Web Conference and is the Semantic Web Track Chair for the 2003 World Wide Web Conference. He is also the coordinator of the EU IST WonderWeb project, and is a principal investigator on several other EU, EPSRC and DARPA funded research projects.

Peter F. Patel-Schneider is a member of technical staff in Bell Labs Research. He received his PhD from the University of Toronto in 1987. Peter was a member of the AI Principles Research Department at AT&T Bell Laboratories from 1988 to 1995, and joined AT&T Labs-Research when AT&T split up. In August 1997, he rejoined Bell Labs. From 1983 to 1988 he worked in the AI research group at Fairchild and Schlumberger. Peter has taught courses at both the University of Toronto and Rutgers University. Peter's research interests centre on the properties and use of Description Logics. He has designed and implemented large sections of CLASSIC, a Description Logic-based knowledge representation system. He designed and implemented DLP, a heavily-optimised prover for expressive Description Logics and propositional modal logics. He is currently involved with the Web Ontology Working Group of the World Wide Web Consortium, designing the OWL language for representing ontologies in the Semantic Web. Peter is also interested in rule-based systems, including more-standard systems derived from OPS as well as newer formalisms such as R++. He designed many of the techniques used in R++ and the R++ translator, and wrote the first several prototype implementations of the R++ translator.

Frank van Harmelen is a professor in knowledge representation & reasoning in the AI department (Faculty of Science) at the Vrije Universiteit Amsterdam. After studying mathematics and computer science in Amsterdam, he moved to the Department of AI in Edinburgh, where he was awarded a Ph.D. in 1989 for his research on meta-level reasoning. While in Edinburgh, he worked with Dr. Peter Jackson on Socrates, a logic-based toolkit for expert systems, and with Prof. Alan Bundy on proof planning for inductive theorem proving. After his Ph.D. research, he moved back to Amsterdam where he worked from 1990 to 1995 in the SWI Department under Prof. Wielinga. He was involved in the REFLECT project on the use of reflection in expert systems, and in the KADS project, where he contributed to the development of the $(ML)^2$ language for formally specifying knowledge-based systems. In 1995, he joined the AI Research Group at the Vrije Universiteit Amsterdam, where he was appointed professor in 2002. His current interests include approximate reasoning, Semantic Web and medical protocols. He has been involved in a multitude of European research projects (REFLECT, CommonKADS, IBROW, Protocure, On-To-Knowledge, SWAP, WonderWeb). He has published three books (on meta-level inference, on knowledge-based systems and on the Semantic Web) and over 100 research papers, most of which can be found on-line at http://www.cs.vu.nl/~frankh.