OWL 2: The Next Step for OWL

Bernardo Cuenca Grau [a], Ian Horrocks [a], Boris Motik [a], Bijan Parsia [b], Peter Patel-Schneider [c], Ulrike Sattler [b]

[a] *University of Oxford, Oxford, UK*
[b] *University of Manchester, Manchester, UK*
[c] *Bell Labs Research, New Jersey, USA*

Abstract

Since achieving W3C recommendation status in 2004, the Web Ontology Language (OWL) has been successfully applied to many problems in computer science. Practical experience with OWL has been quite positive in general; however, it has also revealed room for improvement in several areas. We systematically analyze the identified shortcomings of OWL, such as expressivity issues, problems with its syntaxes, and deficiencies in the definition of OWL species. Furthermore, we present an overview of OWL 2—an extension to and revision of OWL that is currently being developed within the W3C OWL Working Group. Many aspects of OWL have been thoroughly reengineered in OWL 2, thus producing a robust platform for future development of the language.

Key words: Ontologies, Ontology Languages, Semantic Web

1. Introduction

Since the inception of the Semantic Web, the development of languages for modeling ontologies—conceptualizations of a domain shared by a community of users—has been seen as a key task. The initial proposals focused on RDF and RDF Schema; however, these languages were soon found to be too limited in expressive power [18]. The World Wide Web Consortium (W3C) therefore formed the Web Ontology Working Group, whose goal was to develop an expressive language suitable for application in the Semantic Web. The result of this endeavor was the OWL Web Ontology Language, which became a W3C recommendation in February 2004. OWL is actually a family of three language variants (often called *species*) of increasing expressive power: OWL Lite, OWL DL, and OWL Full [28].

The standardization of OWL has sparked the development and/or adaption of a number of reasoners, including FacT++ [37], Pellet [32], RACER [11], and HermiT [26], and ontology editors, including Protégé [1] and Swoop [21]. OWL ontologies are being developed in areas as diverse as e-Science, medicine, biology, geography, astronomy, defense, and the automotive and aerospace industries. OWL is extensively used in the life sciences community, where it has rapidly become a de facto standard for ontology development and data interchange; for example, see BioPAX, [2] NASA's SWEET ontologies, [3] and the

Email addresses:
bernardo.cuenca.grau@comlab.ox.ac.uk (Bernardo Cuenca Grau), ian.horrocks@comlab.ox.ac.uk (Ian Horrocks), boris.motik@comlab.ox.ac.uk (Boris Motik), bparsia@cs.man.ac.uk (Bijan Parsia), pfps@research.bell-labs.com (Peter Patel-Schneider), sattler@cs.man.ac.uk (Ulrike Sattler).

[1] http://protege.stanford.edu/
[2] http://www.biopax.org/
[3] http://sweet.jpl.nasa.gov/ontology/

National Cancer Institute Thesaurus. [4]

Despite the success story surrounding OWL, the numerous contexts in which the language has been applied have revealed some deficiencies in the original design. In Section 2, we present a systematic analysis of problems identified by OWL users and the designers of OWL tools such as editors and reasoners. For example, ontology engineers developing ontologies for biomedical applications have identified significant expressivity limitations of the language. Also, the designers of OWL APIs have identified several practical limitations such as difficulties in parsing OWL ontologies or the inability to check for obvious errors, such as mistyped names.

In response to users' comments and requests, the idea was born to address some of these needs via an incremental revision of OWL, provisionally called OWL 1.1. The initial goal of OWL 1.1 was to exploit recent developments in DL research in order to address some of the expressivity limitations of the language. After extensive discussions at the 2005 OWL Experiences and Directions Workshop, [5] a consensus was reached regarding the new features to be provided by OWL 1.1. This set of new features roughly corresponds to the intersection of what users wanted, what theoreticians said was possible, and what implementors believed was practicable. As the design of OWL 1.1 progressed, it was decided to also take the opportunity to "clean up" the language and its specification, so as to provide a more robust platform for future development.

The development of OWL 1.1 was initially undertaken by an informal group of language users and developers. After the original specification reached a mature state and first implementations were released, the OWL 1.1 proposal was submitted to the W3C as a Member Submission [6] with the intention of using it as a starting point for a new W3C Working Group. The Working Group was officially formed in September 2007. As the work on the new language progressed, the initial Member Submission evolved significantly. Consequently, the Working Group eventually decided in April 2008 to call the new language OWL 2 and so indicate a substantial step in the evolution of the language.

In Section 3, we present the design of OWL 2 and discuss how it addresses the drawbacks of OWL 1 that we identified in Section 2. We discuss different aspects of the language, such as its expressivity, syntax, specification style, and various metalogical features. In Section 4 we discuss the current state of implementation, and conclude in Section 5 with a discussion of possible future extensions. To avoid any ambiguity, we refer to the initial version of OWL as OWL 1 in the rest of this paper.

[4] http://www.nci.nih.gov/

[5] http://www.mindswap.org/2005/OWLWorkshop/

[6] http://www.w3.org/Submission/2006/10/

2. Why Go Beyond OWL 1?

Although, or even perhaps because, OWL 1 has been successful, certain problems have been identified in its design. None of these problems are severe, but, taken together, they indicate a need for a revision of OWL 1. In this section, we discuss what these problems are.

2.1. *Expressivity Limitations*

Practical experience with OWL 1 has shown that OWL 1 DL—the most expressive but still decidable language of the OWL 1 family—lacks several constructs that are often necessary for modeling complex domains [31,30]. As a response, the community of OWL 1 users and application designers developed various patterns for approximating the missing constructs. [7] Since the actual expressive power is missing, these workarounds are often unsound or incomplete with respect to the intended semantics. Furthermore, it is usually difficult, if not impossible, to identify all the cases in which these patterns yield incorrect results. Such patterns are therefore not only cumbersome but also of limited utility, and extending the language with the missing constructs seems to be the only satisfactory solution to the problem.

2.1.1. *Qualified Cardinality Restrictions*

The existential restrictions of OWL 1 DL allow the restriction to be *qualified* with a class; for example, one can define a class such as "persons that have at least one child who is male." Cardinality restrictions, however, cannot be qualified with a class; thus, OWL 1 DL allows for the definition of a person with at least three children, but not of a person with at least three children who are male. Expressing the latter class requires the restriction to be qualified with respect to the class (i.e., male) to which the objects being counted (i.e., children) belong. This

[7] http://www.w3.org/2001/sw/BestPractices/OEP/

feature is typically called a *qualified cardinality restriction* (QCR).

Ontology modelers have repeatedly identified the importance of QCRs in various modeling problems. For example, one may want to define a *quadruped* as an animal with exactly four parts that are legs, or a *medical oversight committee* as a committee that consists of at least five members, of which two are medically qualified, one is a manager, and two are members of the public [30]. In OWL 1, qualified cardinality restrictions are commonly approximated using design patterns discussed in [30,40]; however, these workarounds are often unsound, or incomplete, or both.

At the time the Web Ontology Working Group was designing OWL 1, it was already known that QCRs can be added to the language without affecting its computational properties, both from a theoretical and from an implementation point of view; in fact, QCRs were included in DAML+OIL—a language that served as the basis for OWL 1 [18]. The final decision was, however, not to include this particular feature in the language due to concerns about user understandability.

2.1.2. *Relational Expressivity*

While OWL 1 provides a wide range of constructors for building complex classes, relatively little can be said about properties. Ontology modelers have repeatedly asked for greater relational expressivity (i.e., expressivity about properties), and the lack of this expressivity has been identified as a major impediment for the adoption of OWL 1. We will illustrate this point with two prominent use cases.

Propagation along properties. Applications commonly need to model interactions that are sometimes described as one property "propagating" or being "transitive across" another. Use cases abound in the life sciences domain, where one often needs to describe interactions between locative properties and various kinds of part-whole properties [29]. For example, we might want to assert that an abnormality of a part of an anatomical structure constitutes an abnormality of the structure as a whole [29]. This allows us to draw many useful inferences, such as inferring that a fracture of the neck of the femur is a kind of fracture of the femur, or that an ulcer located in the gastric mucosa is a kind of stomach ulcer. Languages specifically designed for use in life sciences such as OBO [8] and SNOMED [9] commonly provide for such features, even though they typically support a much smaller set of class constructors than OWL 1. Thus, supporting transitive propagation of roles in OWL seemed vital if OWL were to gain widespread acceptance in the life sciences.

Properties of properties. In mereology—the philosophic study of parts and wholes—the *partOf* relation is often specified to be *transitive* (if x is a part of y and y is a part of z, then x is a part of z), *reflexive* (every object is a part of itself), and *asymmetric* (nothing is a part of one of its parts). Many applications that describe complex structures (such as life science and engineering applications) extensively use part-whole relations axiomatized according to these principles. Other types of properties with different axiomatizations are often found in practice (see, e.g., the OBO Relations Ontology [10]), such as locative relations (typically transitive and reflexive), causal relations (typically transitive and irreflexive), and membership relations (typically irreflexive).

At the time OWL 1 DL became a W3C recommendation, it was not known whether the language could be augmented with such features (apart from transitivity) without giving up the decidability of key inference problems, much less whether it would be possible to develop practical reasoning procedures for such extensions. Consequently, such features were excluded from OWL 1. As in the case of QCRs, users have developed modeling patterns intended to approximate the missing features, which has often lead to problems in practice.

2.1.3. *Datatype Expressivity*

OWL 1 provides very limited expressive power for describing classes whose instances are related to concrete values such as integers and strings. In OWL 1, it is possible to express restrictions on datatype properties qualified by a unary datatype. For example, one could state that every British citizen must have a passport number which is an *xsd*:*string*, where the latter is an XML Schema datatype—a unary predicate interpreted as the set of all string values. In OWL 1, however, it is not possible to represent the following kinds of statements:

[8] `http://obo.sourceforge.net/`
[9] `http://www.snomed.org/`
[10] `http://www.obofoundry.org/`

– *restrictions to a subset of datatype values* (e.g., a gale is a wind whose speed is in the range from 34 to 40 knots);
– *relationships between values of data properties on one object* (e.g., a square table is a table whose breadth equals its depth);
– *relationships between values of data properties on different objects* (e.g., people who are older than their boss); or
– *aggregation functions* (e.g., the duration of a process is the sum of the durations of its subprocesses).

Another important limitation of the datatype support in OWL 1 is the lack of a suitable set of built-in datatypes. OWL 1 relies on XML Schema [11] for the list of built-in datatypes. The design of OWL 1, however, did not involve a thorough analysis of which XML Schema datatypes were appropriate for OWL 1. OWL 1 only requires the implementation of *xsd*:*string* and *xsd*:*integer*, and leaves the implementation of other XML Schema datatypes as optional. A subsequent analysis has revealed that datatypes in XML Schema and OWL 1 are used in different ways, so not all datatypes of one formalism are appropriate for the other and vise versa. In particular, many XML Schema datatypes can be difficult to implement in OWL 2. The datatypes *xsd*:*double* and *xsd*:*float* are finite and are thus equivalent to very large disjunctions, which can be a significant source of inefficiency. Furthermore, datatypes that represent time intervals, such as *xsd*:*gDay* and *xsd*:*time*, require the representation of infinite sets of irregular intervals; currently, it is not clear whether and how such datatypes should be implemented in practice.

2.1.4. *Keys*

OWL 1 DL does not provide means for expressing *key constraints* on data properties, which are a core feature of database technologies. For example, in OWL 1 DL it is not possible to state that "US citizens are uniquely identified by their social security number." Such statements can be expressed in OWL 1 Full by means of inverse-functional data properties; however, since no implementations of OWL 1 Full are available (see Section 2.7), there was no practical reasoning support for such statements. Furthermore, no variant of OWL 1 supports *compound key constraints* on either data or object properties, such as that "each address is uniquely identified by its street, street number, postcode, and country."

The lack of keys in OWL 1 has been recognized as an important limitation in expressive power. Unfortunately, adding keys in its full generality to OWL 1 would harm the computational properties of the language and could even lead to undecidability[23].

2.2. *Syntax Issues*

OWL 1 comes with two normative syntaxes: the Abstract Syntax [12] and OWL 1 RDF. [13] The standard also defines an XML syntax, but this syntax is not normative and it has not been widely used, so we do not discuss it here. The Abstract Syntax serves as the actual definition of the language, and it has often been used as basis for the design of OWL 1 APIs. Certain design choices taken in OWL 1 Abstract Syntax, however, have made the syntax confusing for developers, which resulted in the suboptimal design of OWL APIs. Furthermore, both syntaxes have proved difficult to parse correctly. Finally, the relationship between the two syntaxes is rather complex, which causes problems when transforming an ontology from one syntax into the other. We next discuss these problems in more detail.

2.2.1. *Frame-Based Paradigm*

The design of the OWL 1 Abstract Syntax has been heavily influenced by the tradition of frame-based ontology languages. The frame-based paradigm was already familiar to many users, and has proved natural and popular. Consider the OWL 1 DL axiom (1), which declares a class *Tiger* and states that it is a subclass of the class *Cat*:

$$\mathsf{Class}(\mathit{Tiger}\ \mathsf{partial}\ \mathit{Cat}) \tag{1}$$

In a frame-based system, this axiom is usually interpreted as a *frame specification* for the class *Tiger*, which typically acts as a *declaration*—a statement that a class exists in an ontology—and groups all relevant properties of the class in one place.

Although the frame-based paradigm is sometimes useful for modeling, the logical underpinning of OWL 1 is actually provided by the somewhat different paradigm of description logics (DLs). The fundamental modeling concept in DLs is not a frame, but an *axiom*—a logical statement about

[11] http://www.w3.org/TR/xmlschema-2/

[12] http://www.w3.org/TR/owl-semantics/
[13] http://www.w3.org/TR/owl-ref/

the relationships between properties and/or classes in the domain. The following is a subclass axiom that defines an additional property of tigers:

$$\mathsf{SubClassOf}(\mathit{Tiger}\ \mathit{Predator}) \tag{2}$$

The relationship between frames and axioms in the Abstract Syntax has been a major source of confusion. For example, in a frame-based system, one would expect each class or property to be defined using at most one frame; furthermore, such a system would naturally support an operation such as "get all explicitly told superclasses of a class C" that would extract the set of superclasses of C from the frame of C. Since OWL 1 mimics the frame-based paradigm, it should support such an operation; however, since it provides for both frames and axioms, the meaning of such an operation is not clear. For example, given the axioms (1)–(2), according to the frame-based paradigm alone, the result should be only *Cat*, since only this class appears in the frame of *Tiger*. The class *Predator*, however, is also a told superclass of *Tiger*; the only difference is that this relationship has not been defined in the frame for *Tiger*. Thus, according to the axiom paradigm, the intuitively "correct" answer is *Cat* and *Predator*.

These problems have affected the design of OWL 1 APIs. For example, the well-known Manchester OWL API[14] initially tried to faithfully reflect both paradigms. Given axioms (1)–(2), it would return *Cat* as the only superclass of *Tiger*, while the relationship between *Tiger* and *Predator* is made accessible separately as an axiom. This turned out to be confusing for API users since, from the logical point of view, both (1) and (2) just specify an inclusion relationship between two classes.

2.2.2. *Alignment with DL Constructs*

Even though OWL 1 is based on DLs, the constructs of the OWL 1 Abstract Syntax do not completely correspond to the constructs of DLs. Consider the following class definition:

$$\begin{array}{l}\mathsf{restriction}(\mathit{hasParent}\\ \quad \mathsf{someValuesFrom}(\mathit{Person})\\ \quad \mathsf{allValuesFrom}(\mathit{Person}))\end{array} \tag{3}$$

Most DLs allow only one class to appear in a property restriction. For example, all DL implementations known to us translate definitions of the form (3) into the following form:

$$\begin{array}{l}\mathsf{intersectionOf}(\\ \quad \mathsf{restriction}(\mathit{hasParent}\\ \qquad \mathsf{someValuesFrom}(\mathit{Person}))\\ \quad \mathsf{restriction}(\mathit{hasParent}\\ \qquad \mathsf{allValuesFrom}(\mathit{Person})))\end{array} \tag{4}$$

This caused confusion with developers of OWL 1 APIs who, whenever such differences arose, chose to follow the DL structure. Thus, reading and saving an ontology can actually change the structure of the axioms due to mismatches between the API representation and the actual language.

2.2.3. *Determining the Types of Ontology Entities*

Neither the Abstract Syntax nor OWL 1 RDF is fully context-free—that is, an axiom containing a URI often does not contain sufficient information to disambiguate the *type* of ontology entity (i.e., a class, a property, or an individual) that the URI refers to. Consider the following OWL 1 class definition:

$$\begin{array}{l}\mathsf{Class}(\mathit{Person}\ \mathsf{partial}\\ \quad \mathsf{restriction}(\mathit{hasMother}\\ \qquad \mathsf{someValuesFrom}(\mathit{Woman}))\end{array} \tag{5}$$

From this axiom alone, it is not clear whether *hasMother* is a data property and *Woman* is a datatype, or whether *hasMother* is an object property and *Woman* is a class. To disambiguate the syntax, OWL 1 DL relies on a strict separation of the vocabulary into individuals, classes, and data and object properties. The specification of OWL 1 DL, however, does not precisely specify how to enforce this separation at the syntactic level. Thus, whereas the semantics of OWL 1 DL requires strict typing of all names, the syntax does not enforce it, which can prevent certain ontologies from being interpreted. For example, an ontology containing only the axiom (5) seems to be a valid OWL 1 DL ontology; however, the axioms in it are not sufficient to disambiguate the types of the symbols *hasMother* and *Woman*, so the ontology cannot be correctly interpreted. The typing problem is further exacerbated by the fact that the types of ontology entities can be defined in imported ontologies, and the specification of OWL 1 DL provides no guidance on how to disambiguate the types in such cases and how to resolve potential conflicts. OWL 1 tools have dealt with this problem in ad hoc ways, which has adversely affected interoperability.

This problem is made even more complex by the fact that names for ontology entities can be introduced without any prior announcement. This makes detecting trivial syntactic problems difficult, such as

[14] `http://sourceforge.net/projects/owlapi/`

typographical errors in entity names. Consider, for example, the following two axioms:

$$\mathsf{DisjointClasses}(\mathit{Animal}\ \mathit{Plant}) \quad (6)$$
$$\mathsf{SubClassOf}(\mathit{Human}\ \mathit{Animla}) \quad (7)$$

In axiom (7), instead of *Animal*, the user has inadvertently typed *Animla*, which effectively introduces a new class. Such errors are quite difficult to detect in OWL 1: since there is no explicit statement that an entity exists, axioms (6) and (7) constitute a valid OWL 1 ontology.

2.2.4. *Problems with OWL 1 RDF*

The vast majority of OWL 1 ontologies have been written in the OWL 1 RDF syntax; this syntax has, however, shown itself to be quite difficult to use in practice [5]. The main difficulty is that RDF represents everything using triples, which specify relationships between pairs of objects. In contrast, many OWL 1 constructs cannot be represented using triples without the introduction of new objects. For example, to represent in OWL 1 RDF that class A is the union of B and C, the following collection of triples is required, in which the union operator is represented by a blank node $\_{:}x1$ that encodes a list of objects:

$$\langle A, \mathit{owl{:}unionOf}, \_{:}x1\rangle \quad (8)$$
$$\langle \_{:}x1, \mathit{rdf{:}first}, B\rangle \quad (9)$$
$$\langle \_{:}x1, \mathit{rdf{:}rest}, \_{:}x2\rangle \quad (10)$$
$$\langle \_{:}x2, \mathit{rdf{:}first}, C\rangle \quad (11)$$
$$\langle \_{:}x2, \mathit{rdf{:}rest}, \mathit{rdf{:}nil}\rangle \quad (12)$$

This makes OWL 1 RDF ontologies difficult to read and process.

Another problem with OWL 1 RDF is that the triples corresponding to a single OWL 1 construct need not be grouped together, and may even be split across multiple documents. This further exacerbates the typing problems discussed in Section 2.2.3. In order to deal with this situation, all OWL 1 RDF parsers known to us first load all RDF triples into memory, analyze them, and then produce appropriate OWL 1 axioms. This is clearly inefficient, and it limits the scalability of OWL 1 systems.

Moreover, the OWL 1 RDF and OWL 1 Abstract Syntax do not correspond completely. For example, the axioms

$$\mathsf{Class}(\mathit{Human}\ \mathsf{partial}\ \mathit{Animal}) \quad (13)$$
$$\mathsf{SubClassOf}(\mathit{Human}\ \mathit{Animal}) \quad (14)$$

are both equivalent to the following RDF triple:

$$\langle \mathit{Human}, \mathit{rdfs{:}subClassOf}, \mathit{Animal}\rangle \quad (15)$$

The OWL 1 specification provides only a mapping from the Abstract Syntax into OWL RDF, but not the converse. As a result, according to the OWL 1 specification, an RDF graph $\mathcal{G}$ is an OWL 1 DL ontology if there exists an ontology $\mathcal{O}$ in Abstract Syntax such that the result of the normative transformation of $\mathcal{O}$ into triples is precisely $\mathcal{G}$. This makes checking whether $\mathcal{G}$ is an OWL 1 DL ontology very hard in practice: one essentially needs to examine all "relevant" ontologies $\mathcal{O}$ in abstract syntax and check whether the normative transformation of $\mathcal{O}$ into RDF yields precisely $\mathcal{G}$. This is clearly inefficient, so different OWL 1 tools have applied different ad hoc optimizations, which is a significant source of incompatibility between tools.

2.3. *Metamodeling*

In many practical cases, the distinction between classes and individuals is not clear-cut. Consider the statements that "Harry is an Eagle" and "Eagles are an endangered species." The first statement can be modeled in OWL 1 by asserting the individual *Harry* to be an instance of the class *Eagle*. The second statement, however, talks about eagles as a species, and not as a set of all living eagles; therefore, it should be modeled by stating the individual *Eagle* to be an instance of the class *EndangeredSpecies*. Hence, *Eagle* plays the role of a class in one and of an individual in another context. This style of modeling is often called *metamodeling*.

The W3C OWL Working Group acknowledged the importance of metamodeling in practice; however, at the time OWL 1 was designed, metamodeling had not been widely considered in DL research. Furthermore, as we discussed in Section 2.2.3, the sets of names used for classes, properties, and individuals must be disjoint in OWL 1 Lite and OWL 1 DL ontologies. Therefore, metamodeling is possible only in the OWL 1 Full species of OWL 1. This is problematical, as OWL 1 Full has not been widely implemented. Moreover, in [24] it was shown that the style of metamodeling used in OWL 1 Full leads to the undecidability of standard reasoning problems, making it unlikely that this style of metamodeling will be made available in OWL 1 tools. Users are therefore forced to choose between metamodeling, which may be needed in their application, and tool support for ontology development.

2.4. *Imports and Versioning*

OWL 1 provides a basic mechanism that allows an ontology $\mathcal{O}$ to *import* another ontology $\mathcal{O}'$, and thus gain access to all entities and axioms in $\mathcal{O}'$. This can be understood as a form of "virtual" inclusion: semantically, every model of $\mathcal{O}$ must also satisfy all axioms of $\mathcal{O}'$; however, the two ontologies are kept physically separate. Imports in OWL 1 DL are "by name and location": the ontology $\mathcal{O}$ is required to contain a URI pointing to the location of $\mathcal{O}'$, and this location should match with the name of $\mathcal{O}'$.

The coupling of names and locations seems appropriate when ontologies are published at a fixed location on the Web. Applications, however, often use OWL 1 ontologies off-line, by loading them from local ontology repositories. Ontologies are also often moved between locations. Thus, in many realistic use cases, the location of an ontology does not correspond with its name. Tools have dealt with this in various ways. Some tools have stayed true to the official OWL 1 specification, thus requiring users to manually adjust the names of ontologies and the import relations between them when ontologies are moved around. This approach is often cumbersome, particularly when an ontology depends on a large number of other ontologies; therefore, many tools have sacrificed compatibility with the official specification and have provided ad hoc caching and location redirection mechanisms.

These problems become even more acute when one needs to maintain several different versions of the same ontology. The specification of OWL 1 provides no guidelines on how to handle such cases.

2.5. *Annotations*

OWL 1 ontologies and entities can be assigned annotations, which are pieces of extra-logical information describing the ontology or entity. Annotations in OWL 1 are written using *annotation properties*. OWL 1 provides several built-in annotation properties, some of which are inherited from RDF: *owl*:*versionInfo*, *rdfs*:*label*, *rdfs*:*comment*, *rdfs*:*seeAlso*, and *rdfs*:*isDefinedBy*. Ontology engineers can also create their own annotation properties and use them in their ontologies. OWL 1 Full does not put any constraints on the usage of annotation properties in an ontology. OWL 1 DL, however, disallows the usage of annotation properties in OWL 1 DL axioms; for example, it is not possible in OWL 1 DL to define a subproperty or to place a domain or a range constraint on an annotation property. To harmonize their treatment between OWL 1 DL and RDF, annotation properties are given semantics as binary relations in the interpretation domain. In this sense, annotation properties are treated semantically in a very similar way to object properties.

Because they cannot be used in axioms, users typically expect annotations to be extra-logical constructs: adding or removing annotations should not affect the set of consequences derivable from an ontology. Therefore, providing an explicit semantics to annotations in OWL 1 DL is rather counterintuitive, as it makes annotation properties look like "poor man's properties." All OWL 1 DL tools we know of simply ignore the formal semantics of annotations and thus are not strictly compliant with the official specification of OWL 1 DL. Access to annotations is typically provided in such tools through nonlogical methods such as various API extensions.

In addition to this basic issue, the annotation system of OWL 1 has been found to be inadequate for many applications. In particular, OWL 1 does not allow axioms to be annotated, which can be necessary, for example, to represent provenance information (e.g., who wrote a particular axiom) or for language extensions (e.g., to represent the confidence in the validity of axioms). Finally, users of OWL 1 DL have often complained about the inability to define domains and ranges of annotation properties, or to use annotation properties in property hierarchies (e.g., to say that the range of the *proteinID* annotation property is string).

2.6. *OWL Semantics*

OWL 1 DL was designed, on the one hand, as a notational variant of the expressive description logic $\mathcal{SHOIN}(\mathbf{D})$ [17]; on the other hand, it was very important for OWL 1 to be compatible with existing Semantic Web languages such as RDF. Semantic differences between $\mathcal{SHOIN}(\mathbf{D})$ and RDF made it difficult to satisfy both requirements. The solution to this problem chosen by the OWL Working Group was to provide two coexisting semantics for OWL 1. The first is a direct model-theoretic semantics based on the semantics of $\mathcal{SHOIN}(\mathbf{D})$, and is the normative semantics for OWL 1 Lite and OWL 1 DL. The second is an extension of the RDF model theory [12], and is applicable to OWL 1 ontologies writ-

ten in RDF. The practical experience with OWL 1 has identified a number of problems in each of these semantics alone, as well as in the difficult marriage between the two semantics.

2.6.1. *OWL 1 DL Semantics*

The description logic $\mathcal{SHOIN}(\mathbf{D})$ has been extensively investigated in the literature, and its expressivity and computational properties are well-known [36,19]. The OWL 1 DL specification provides an explicit (and quite complex) definition of the semantics, which does not straightforwardly correspond to $\mathcal{SHOIN}(\mathbf{D})$. This caused a number of problems regarding presentation and understandability. Tool implementors are thus unnecessarily burdened with the effort of establishing a correspondence between OWL 1 DL and $\mathcal{SHOIN}(\mathbf{D})$ in order to reuse known reasoning algorithms.

A source of confusion is the ability to use unnamed individuals in OWL 1 facts (this feature was inspired by RDF *blank nodes*). For example, one can assert an individual *John* to be a *friendOf* a *friendOf* an individual *Paul* without naming the intervening individual. Unnamed individuals are not directly available in $\mathcal{SHOIN}(\mathbf{D})$, and it is not obvious how to simulate them.

Another source of confusion is due to the frame-based constructs of OWL 1 (see Section 2.2.1 for an in-depth discussion). $\mathcal{SHOIN}(\mathbf{D})$ is purely axiom-based and it provides no frame-like features. The frame-like axioms of OWL 1 DL thus need to be translated into $\mathcal{SHOIN}(\mathbf{D})$ for the purpose of reasoning, which is unnecessarily complex.

2.6.2. *RDF-Compatible Semantics*

OWL 1 ontologies written as RDF graphs can be interpreted using an extension of the RDF semantics. This semantics, however, has proved to be extremely complex to understand, use, and implement.

First, the RDF-compatible semantics of OWL 1 is not a standard first-order semantics; rather, it is fully reified: classes and properties are assigned extensions indirectly, by first mapping them into elements of the domain of discourse and then assigning an extension to the domain elements. This adds an additional level of complexity in understanding the logical consequences of an ontology.

Second, the RDF-compatible semantics of OWL 1 includes so-called *comprehension principles* that enforce the existence of an individual in the interpretation domain for each class and property that can be formed using the vocabulary of the ontology. For example, if A and P are a class and a property, respectively, and $\mathcal{I}$ is an RDF interpretation, then $\mathcal{I}$ must contain domain elements o_A and o_P for A and P themselves, as well as for all the infinitely many classes that can be built from A and P, such as $o_{\exists P.A}$, $o_{\exists P \exists P.A}$, and so on. As a consequence, the domain of every OWL 1 Full interpretation must be infinite. Furthermore, the comprehension principles make it difficult to verify whether the RDF-compatible semantics of OWL 1 contains a built-in contradiction; that is, it may even be the case that even an empty RDF graph is inconsistent under the RDF-compatible semantics.

Third, even if the semantics were consistent, the implementation of a complete reasoner would still be infeasible, as checking satisfiability of RDF graphs interpreted under the RDF-compatible semantics of OWL 1 is undecidable due to the mixing of logical and metalogical symbols [24].

Fourth, the RDF-compatible semantics of OWL 1 is not robust under language extensions: if OWL 1 were extended to cover all of first-order logic, the RDF-compatible semantics would become inconsistent due to logical paradoxes [27].

2.6.3. *Relating the DL and the RDF Semantics*

A key idea behind the two semantics of OWL 1 was that they should be "equivalent" on OWL 1 DL and OWL 1 Lite ontologies. This requirement, however, has been found difficult to realize, so it was subsequently weakened. The OWL 1 DL specification contains a proof that, for any two OWL 1 DL ontologies $\mathcal{O}$ and $\mathcal{O}'$ written in OWL 1 Abstract Syntax, if $\mathcal{O}$ entails $\mathcal{O}'$, then the translation of $\mathcal{O}$ into RDF entails the translation of $\mathcal{O}'$ into RDF as governed by the (extended) RDF semantics.

Furthermore, maintaining this compatibility becomes increasingly problematical with increasing expressive power. It has been shown that it would be impossible to maintain compatibility with the OWL 1 RDF semantics if OWL 1 were extended to the expressive power of first-order logic [27]. The coexistence of the two semantics is therefore clearly a dangerous choice with respect to the possible extension of OWL 1 DL: even if these extensions would be decidable under either semantics, the two semantics might no longer be equivalent.

2.7. *OWL 1 Full*

OWL 1 Full is the most expressive variant of OWL 1. Each RDF graph is a syntactically valid OWL 1 Full ontology; that is, in contrast to OWL 1 DL, in OWL 1 Full there are no syntactic restrictions on the usage of the built-in OWL 1 vocabulary and the vocabulary elements defined in the ontology. The semantics of OWL 1 Full is given by the RDF-compatible semantics of OWL 1.

As discussed in the previous section, the basic reasoning problems for the RDF-compatible semantics are undecidable. Furthermore, the free usage of vocabulary adds an additional source of undecidability; for example, OWL 1 Full does not enforce the well-known restrictions needed for decidability, such as using only simple roles in number restrictions [20]. To the best of our knowledge, no complete implementation of OWL 1 Full currently exists, and it is not clear whether OWL 1 Full can be implemented in practice.

2.8. *OWL 1 Lite*

Although it is decidable, reasoning in OWL 1 DL is of a high worst-case computational complexity (NExpTime-complete [36]). Therefore, the OWL Working Group considered it important to define a fragment of OWL 1 DL with more tractable inference problems and that was easier to understand and use. To this purpose, the W3C OWL Working Group defined the OWL 1 Lite subset of OWL 1 DL. OWL 1 Lite is a syntactic subset of OWL 1 DL that excludes constructors that some thought to be difficult to use and/or lead to high computational complexity. In particular, OWL 1 Lite does not allow the use of union and complement constructors in class descriptions, it limits descriptions in the scope of a quantifier to be class names, it disallows the oneOf constructor (i.e., nominals), and it limits the numbers in cardinality restriction to 0 and 1 only.

Unfortunately, even though OWL 1 Lite seems to be much simpler than OWL 1 DL, most of the complexity of OWL 1 DL can be captured due to implicit negations in axioms. For example, the following two OWL 1 Lite axioms implicitly define the class C as the negation of D:

$$\begin{array}{l}\mathsf{Class}(C \text{ complete} \\ \quad \mathsf{restriction}(P \\ \qquad \mathsf{allValuesFrom}(owl{:}Nothing)))\end{array} \tag{16}$$

$$\begin{array}{l}\mathsf{Class}(D \text{ complete} \\ \quad \mathsf{restriction}(P \\ \qquad \mathsf{someValuesFrom}(owl{:}Thing)))\end{array} \tag{17}$$

In fact, of all the OWL 1 DL constructors, only nominals and cardinality restrictions with cardinality larger than one cannot be captured in OWL 1 Lite, thus making the language equivalent to the description logic $\mathcal{SHIF}(\mathbf{D})$. Hence, from a users' perspective, OWL 1 Lite is confusing since the available modeling constructs do not correspond to the expressivity of the language. Moreover, from a computational perspective, basic reasoning problems are only slightly less complex for OWL 1 Lite than for OWL 1 DL (ExpTime- instead of NExpTime-complete), so the language remains highly intractable.

2.9. *Species Validation*

Species validation is the problem of determining whether an OWL 1 ontology is in OWL 1 Lite, OWL 1 DL, or OWL 1 Full. This seemingly trivial task turned out to be rather problematic in OWL 1.

2.9.1. *OWL 1 Lite or OWL 1 DL?*

The OWL 1 specification defines OWL 1 Lite in terms of a fragment of the OWL 1 DL Abstract Syntax: an OWL 1 DL ontology $\mathcal{O}$ in Abstract Syntax is also an OWL 1 Lite ontology if and only if it can be generated using the OWL 1 Lite grammar, as given in the normative documents.

In the case of ontologies written in OWL 1 RDF, the lack of a direct mapping between OWL 1 RDF and Abstract Syntax makes species validation problematic. If $\mathcal{O}$ is given in the OWL 1 RDF syntax, then it is an OWL 1 Lite ontology if and only if *there exists* an ontology $\mathcal{O}'$ in the OWL 1 Lite fragment of the Abstract Syntax such that the translation of $\mathcal{O}'$ into triples is precisely $\mathcal{O}$. Similarly, an ontology $\mathcal{O}$ in the OWL 1 RDF syntax is an OWL 1 DL ontology if and only if there exists an ontology $\mathcal{O}'$ in the Abstract Syntax such that the translation of $\mathcal{O}'$ into triples is precisely $\mathcal{O}$.

Therefore, determining whether an ontology in RDF is an OWL 1 Lite or an OWL 1 DL ontology becomes very hard in practice since it involves "guessing" ontologies in Abstract Syntax and checking whether their normative transformation into RDF yields precisely the original ontology. In practice, validating parsers are usually based on

informal guidelines [4] that are not an official part of the OWL 1 specification.

Moreover, as we already discussed, the expressive power of OWL 1 Lite corresponds to the description logic $\mathcal{SHIF}(\mathbf{D})$. Now consider an ontology $\mathcal{O}$ that, for example, uses the unionOf construct, but actually corresponds to $\mathcal{SHIF}(\mathbf{D})$. Technically, $\mathcal{O}$ is an OWL 1 DL ontology; however, an OWL 1 Lite ontology $\mathcal{O}'$ exists that is equivalent to $\mathcal{O}$. Any reasoner for $\mathcal{O}'$ could handle $\mathcal{O}$ without any problems (the complexity of the reasoner is defined by the actual DL fragment). Thus, the distinction between $\mathcal{O}$ and $\mathcal{O}'$ seems quite artificial, and classifying $\mathcal{O}$ as being "harder" than $\mathcal{O}'$ is not intuitive.

2.9.2. *Species Validation and Imports*

Suppose that $\mathcal{O}_1$ is an OWL 1 ontology that imports another OWL 1 ontology $\mathcal{O}_2$; logically, we thus get a new ontology $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$. We identify two basic requirements concerning the robustness of the imports mechanisms regarding species validation.

First, membership in a certain species of OWL 1 should ideally be preserved under imports; for example, if $\mathcal{O}_1$ and $\mathcal{O}_2$ are in OWL 1 DL, then it is intuitive to expect $\mathcal{O}$ to be in OWL 1 DL as well.

Second, if $\mathcal{O}_1$ and $\mathcal{O}_2$ are written in different species, one would expect $\mathcal{O}$ to be in the most expressive one; for example, if $\mathcal{O}_1$ is in OWL 1 Lite and $\mathcal{O}_2$ is in OWL 1 DL, we would expect $\mathcal{O}$ to be in OWL 1 DL.

Unfortunately, imports can interact with OWL 1 species in a quite unpredictable and unintuitive way. As a result, none of these requirements are satisfied.

As an example of a violation of these requirements, suppose that $\mathcal{O}_1$ is an OWL 1 DL ontology that defines a property P as transitive. Assume also that the ontology $\mathcal{O}_2$ is an OWL 1 DL ontology that uses P in a cardinality restriction, but it does not define P as transitive. The ontology $\mathcal{O}$ is no longer in OWL 1 DL since P is not a simple property (due to transitivity) and it is used in a cardinality restriction. Therefore, $\mathcal{O}$ is an OWL 1 Full ontology.

Furthermore, the following example demonstrates a surprising fact: $\mathcal{O}_1$ and $\mathcal{O}_2$ can be in OWL 1 Full, while $\mathcal{O}$ is in OWL 1 Lite. Suppose that $\mathcal{O}_1$ contains just the following triples:

$$\langle A, \mathit{rdfs{:}subClassOf}, B\rangle \tag{18}$$

$$\langle A, \mathit{rdf{:}type}, \mathit{owl{:}Class}\rangle \tag{19}$$

Furthermore, $\mathcal{O}_2$ contains just the following triples:

$$\langle B, \mathit{rdfs{:}subClassOf}, A\rangle \tag{20}$$

$$\langle B, \mathit{rdf{:}type}, \mathit{owl{:}Class}\rangle \tag{21}$$

The ontology $\mathcal{O}_1$ is in OWL 1 Full because it does not contain the triple (21) specifying the type of B; similarly, $\mathcal{O}_2$ is is OWL 1 Full because it misses the triple (19). The union of these ontologies, however, contains all necessary triples, so it is in OWL 1 Lite.

3. The Design of OWL 2

To address the problems with OWL 1 that we identified in Section 2, the design of OWL 2 has departed from that of OWL1 in several ways.

3.1. *Increasing Language Expressivity*

The drawbacks of OWL 1 regarding expressivity identified in Section 2.1 have long been recognized in the DL community, and a significant amount of research has been devoted to finding possible solutions. The cumulative results of this work are embodied in the DL $\mathcal{SROIQ}$ [16], which is strictly more expressive than $\mathcal{SHOIN}$. This is evidenced by an increase in the computational complexity of the basic reasoning problems: reasoning in $\mathcal{SROIQ}$ is 2NExpTime-complete [22], whereas in $\mathcal{SHOIN}$ it is NExpTime-complete [36].

Although reasoning in $\mathcal{SROIQ}$ is harder than in $\mathcal{SHOIN}$, the tableaux-based reasoning algorithm for $\mathcal{SROIQ}$ [16] follows the same principles as the one for $\mathcal{SHOIN}$. If the new features are not used, then the new reasoning algorithm behaves just like the old algorithm for $\mathcal{SHOIN}$. Furthermore, the source of added complexity is well understood [22], and it seems realistic to expect that the complexity increase will not occur on typical practical problems. Finally, existing reasoners can and have been easily extended to $\mathcal{SROIQ}$. Thus, $\mathcal{SROIQ}$ seems to provide a good logical underpinning for OWL 2 from both a theoretical and a practical perspective.

3.1.1. *Qualified Number Restrictions*

Even while OWL 1 was being designed, it was known that QCRs could have been added to the language without any theoretical or practical problems: the resulting logics are still decidable and have been successfully implemented in practical reasoning systems. Moreover, as mentioned in Section 2.1.1, QCRs have been supported in DAML+OIL—a predecessor of OWL. Thus, qualified number restrictions are not really a novel feature of either

$\mathcal{SROIQ}$ or of DL-based ontology languages, and were incorporated into OWL 2 in the obvious way.

3.1.2. *Relational Expressivity*

As mentioned in Section 2.1, a major drawback of OWL 1 is the limited expressivity regarding properties. This is addressed in $\mathcal{SROIQ}$ and OWL 2 by the addition of *complex property inclusion axioms*, which significantly increase the relational expressivity of the language. In particular, they provide for the propagation of one property along another. For example, axiom (22) states that, if a contains b and b has a part c, then a also contains c:

$$\mathsf{SubPropertyOf}(\mathsf{PropertyChain}(\mathit{contains}\ \mathit{hasPart})\ \mathit{contains}) \tag{22}$$

If such axioms are used in an unrestricted way, they easily make the logic undecidable. To achieve decidability, $\mathcal{SROIQ}$ (and hence OWL 2) imposes a *regularity restriction* on such axioms: roughly speaking, it must be possible to arrange all properties in a total ordering $\succ$ such that, for each property p on the left-hand side of a subproperty axiom, p is either equal to or a $\succ$-predecessor of the property on the right-hand side of the axiom. Although this might seem rather complex, it merely formalizes a simple restriction that complex subproperty axioms should not define properties in a cyclic way. For example, although (22) by itself does not cause problems, it should not be simultaneously used with the following axiom, which states that if b is a part of a and b contains c, then c is also a part of a:

$$\mathsf{SubPropertyOf}(\mathsf{PropertyChain}(\mathit{hasPart}\ \mathit{contains})\ \mathit{hasPart}) \tag{23}$$

The axioms (22) and (23) together introduce a cyclic dependency between *contains* and *hasPart*, which can easily lead to problems with decidability.

In addition to complex property inclusion axioms, properties in $\mathcal{SROIQ}$ and OWL 2 can be transitive, reflexive, irreflexive, symmetric, and/or asymmetric, and pairs of properties can be made disjoint. Thus, OWL 2 provides many of the features necessary for applying the principles of mereology.

3.1.3. *Increasing Datatype Expressivity*

OWL 2 significantly extends the set of built-in datatypes of OWL 1 by reusing certain datatypes from XML Schema. Thus, OWL 2 now supports *owl*:*boolean*, *owl*:*string*, *xsd*:*integer*, *xsd*:*dateTime*, *xsd*:*hexBinary*, and a number of datatypes derived from these by placing various restrictions on them. In addition, *xsd*:*decimal*, *xsd*:*double*, and *xsd*:*float* will most likely be complemented with *owl*:*real*—a datatype interpreted as the set of all real numbers.

OWL 2 also provides a *datatype restriction* construct, which allows new datatypes to be defined by restricting the built-in datatypes in various ways. For example, the following expression defines a new datatype by specifying a lower bound of 18 on the XML Schema datatype *xsd*:*integer*:

$$\mathsf{DatatypeRestriction}(\mathit{xsd{:}integer}\ \mathit{xsd{:}minInclusive}\ 18) \tag{24}$$

It is worth mentioning, however, that the Working Group has still not made a final decision on the exact set of datatypes that the language will support. Therefore, some of the information in this section may become obsolete in the future.

3.1.4. *Keys*

As mentioned in Section 2.1.4, keys are important in many applications. Extending DL-based languages such as OWL 2 with keys, however, poses both theoretical and practical problems [23]. Therefore, the Working Group has decided to include a more restricted variant of keys that can be useful in practice as well as relatively easy to implement, commonly known as *easy keys*.

OWL 2 thus provides for *key axioms* of the form $\mathsf{HasKey}(CP_1 \dots P_n)$, which state that the object or data properties P_i are keys for (named) instances of the class C—that is, no two (named) instances of C can coincide on the values of all P_i.

For example, the following OWL 2 axiom states that persons are uniquely identified by their social security number:

$$\mathsf{HasKey}(\ \mathit{Person}\ \mathit{hasSSN}\ ) \tag{25}$$

The following assertions state that two individuals—*PSmith* and *PeterSmith*—have the same social security number:

$$\mathsf{PropertyAssertion}(\mathit{PSmith}\ \mathit{hasSSN}\ \text{“123-45-6789”}) \tag{26}$$

$$\mathsf{PropertyAssertion}(\mathit{PeterSmith}\ \mathit{hasSSN}\ \text{“123-45-6789”}) \tag{27}$$

Since the individuals *PSmith* and *PeterSmith* are known by name, axioms (25)–(27) entail that *PSmith* and *PeterSmith* are the same individuals:

$$\mathsf{SameIndividual}(\mathit{PSmith}\ \mathit{PeterSmith}) \tag{28}$$

Unlike the general keys, easy keys are not applied to individuals not known by name. For example, the following axiom states that *Jane* is connected through *marriedTo* to an individual α that is an instance of *Man* and that has "123-45-6789" as the value of *hasSSN*:

```
ClassAssertion(
    SomeValuesFrom(marriedTo
        IntersectionOf(
            Man
            HasValue(hasSSN "123-45-6789")
        )
    )
    Jane )
```
(29)

Even though α has the same value for *hasSSN* as, say, *PSmith*, individual α is not known by name. Therefore, the key axiom (25) is not applicable to α so, consequently, axioms (25), (26), and (29) do not entail the following assertion:

$$\mathsf{ClassAssertion}(\mathit{Man}\ \mathit{PSmith}) \tag{30}$$

Thus, the main drawback of easy keys is that they can only produce consequences about explicit data and are thus relevant mainly for query answering, whereas the general variant of keys [23] can also affect the subsumption hierarchy between classes. The main benefits of easy keys are that adding them to OWL 2 do not increase the worst-case complexity of reasoning, and that implementing them in the existing reasoners is relatively straightforward.

3.1.5. *Extensions under Discussion*

The Working Group is currently discussing whether to extend OWL 2 with *n-ary datatypes*, such as the binary datatype *equal*. Such datatypes could then be used to compare values of data properties for a given object. For example, the following expression defines the class of objects whose width is equal to their height:

$$\mathsf{AllValuesFrom}(\mathit{width}\ \ \mathit{height}\ \ \mathit{equal}) \tag{31}$$

Unfortunately, ever since concrete datatypes were introduced in [2], it has been known that comparing values of data properties for different objects (e.g., defining the class of people who have a child who is older than their spouse) easily leads to undecidability, unless the concept language is significantly weakened. Similarly, aggregate functions can be combined only with very simple DLs if decidability is desired [3], so it was therefore decided not to include these expressive features into OWL 2.

3.2. *The MOF Metamodel for OWL 2*

To solve the problems identified in Section 2.2, the structure of OWL 2 ontologies has been unambiguously specified using OMG's Meta-Object Facility (MOF). [15] MOF is a well-known metalanguage, and has been extensively used for specification of other languages. The MOF metamodel—also called the *structural specification*—of OWL 2 is presented in the documents using the Unified Modeling Language (UML).

The classes of the MOF metamodel describe the canonical structure of OWL 2 ontologies in a way that is independent of the syntax used to serialize the ontologies. The specification consists of 22 UML class diagrams. For example, Figure 1 shows the UML diagram that defines an ontology as consisting of a set of axioms and of a set of annotations; furthermore, it shows that an ontology can import a set of ontologies and that each axiom can contain a set of annotations.

OWL 2 axioms are defined as subclasses of the abstract class Axiom. For example, Figure 2 shows the diagram that defines the structure of class axioms, where the axiom EquivalentClasses is defined as taking a set of classes.

The MOF metamodel for OWL 2 can be seen as analogous to the Document Object Model (DOM) specification [16] for XML. It unambiguously specifies what OWL 2 ontologies are in terms of their structure and thus makes the specification precise. Moreover, it can be taken as a foundation for the design of an OWL 2 API. By committing to a well-known metamodel, APIs of different developers should be interoperable, which will reduce the burden on application developers.

The MOF metamodel of OWL 2 also defines the notion of *structural equivalence*, which determines whether two ontology components are considered to be the same. For example, the following two OWL 2 axioms are not identical; they are, however, structurally equivalent, since the EquivalentClasses axiom consists of a set of classes in which the order of the elements is not relevant:

$$\mathsf{EquivalentClasses}(\mathit{Person}\ \mathit{Human}) \tag{32}$$
$$\mathsf{EquivalentClasses}(\mathit{Human}\ \mathit{Person}) \tag{33}$$

Structural equivalence is important for the definitions of the functionality of OWL 2 APIs and ser-

[15] `http://www.omg.org/mof/`
[16] `http://www.w3.org/DOM/`

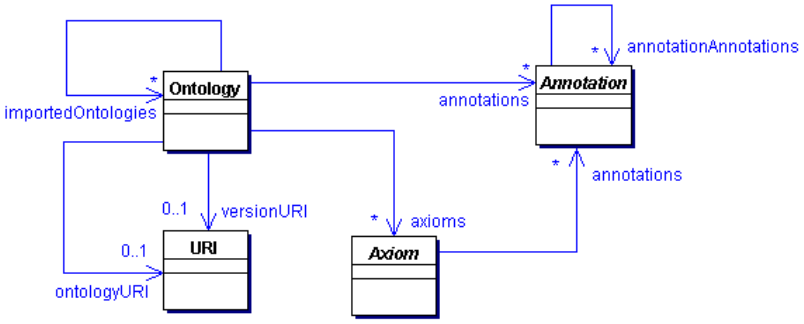

Fig. 1. Object Model of OWL 2 Ontologies

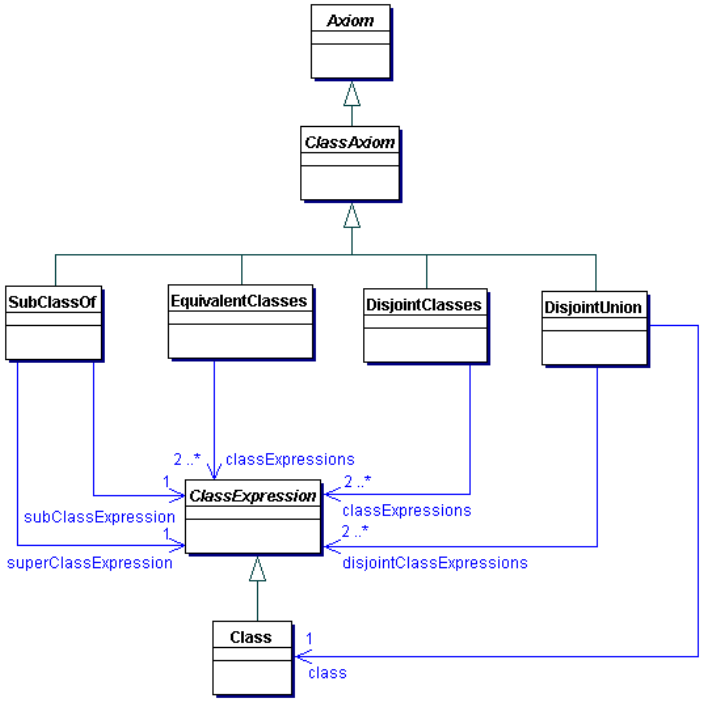

Fig. 2. Class Axioms in OWL 2

vices. For example, OWL 2 reasoners might support incremental addition and deletion of axioms. By relying on the notions of structural equivalence of axioms and sets of axioms, the semantics of the addition and deletion operations can be easily defined in terms of standard set operations on sets of axioms.

3.3. *Typing and Declarations*

To address the problems described in Section 2.2.3 regarding disambiguation of ontology types, OWL 2 introduces the notion of declarations. All entities can, and sometimes even must, be declared in an OWL 2 ontology. A declaration for an entity with a URI u in an ontology $\mathcal{O}$ serves two purposes:

- It states that u is part of the vocabulary of $\mathcal{O}$; for example, an ontology editor can use declarations to implement functions such as "Add New Class."
- It associates with u an entity type—that is, it says whether u is a class, datatype, object property, data property, annotation property, or an individual.

For example, the existence of the classes *Plant* and *Animal* (see axioms (6)–(7)) can be stated in the following way:

$$\mathsf{Declaration}(\mathsf{Class}(\mathit{Plant})) \tag{34}$$

$$\mathsf{Declaration}(\mathsf{Class}(\mathit{Animal})) \tag{35}$$

Declarations are nonlogical statements in the sense that they do not directly affect the semantics; they affect the semantics only indirectly by properly typing the ontology's vocabulary.

Certain declarations are required for an ontology to be syntactically valid. For example, if an ontology contains an axiom

$$\mathsf{SubPropertyOf}(P \quad Q), \tag{36}$$

then P and Q must be declared; otherwise, it would not be possible to tell whether they are object or data properties. In OWL 2, however, a URI u can be used as an individual in $\mathcal{O}$ even if it is not declared as an individual in $\mathcal{O}$. This is so because it is always possible by looking at the syntax of an OWL 2 ontology to distinguish the usages of a URI as a class or an individual.

OWL 2 imposes certain typing constraints on ontologies. Roughly speaking, the sets of object, data, and annotation properties, as well as the sets of classes and datatypes in an OWL 2 ontology must be mutually disjoint. Note that these restrictions are weaker than in OWL 1 DL; for example, a URI u can be used in an OWL 2 ontology to refer to both a class and an individual.

The type of a URI u in an ontology $\mathcal{O}$ is determined by considering all the declarations in $\mathcal{O}$ and the ontologies imported (directly or indirectly) into $\mathcal{O}$. Also, the imported ontologies can be written in different syntaxes. This makes parsing $\mathcal{O}$ somewhat difficult, as it requires two passes through $\mathcal{O}$ and all imported ontologies—one to identify the types of URIs and one to actually construct the axioms of $\mathcal{O}$. The intended result, however, has been precisely described in OWL 2 by means of a *canonical parsing process*, which removes any ambiguity as to what constitutes normative behavior.

Finally, OWL 2 provides a notion of *declaration consistency*: an ontology $\mathcal{O}$ has consistent declarations if each URI u occurring in $\mathcal{O}$ in position of an entity with a type T is declared in $\mathcal{O}$ as having type T. Declaration consistency is not a prerequisite for syntactic validity: an OWL 2 ontology can be used even if its declarations are not consistent. A declaration consistency check, however, can be very useful for detecting trivial errors caused, for example, by mistyping the names of ontology entities.

3.4. *Metamodeling with Punning*

The fact that OWL 2 typing constraints are more permissive than in OWL 1 allows for a form of metamodeling in OWL 2 that addresses some of the limitations described in Section 2.3. For example, the following axioms state the facts that eagles are an endangered species, and that *Harry* is an eagle:

$$\mathsf{ClassAssertion}(\mathit{Eagle}\ \mathit{Harry}) \tag{37}$$
$$\mathsf{ClassAssertion}(\mathit{Endangered}\ \mathit{Eagle}) \tag{38}$$

Note that the symbol *Eagle* is used in (37) as a class, whereas in (38) it is used as an individual.

The semantics of metamodeling in OWL 2 corresponds to the *contextual semantics* defined in [24]: the usage of *Eagle* as a class is completely unrelated to its usage as an individual. In fact, the semantics can be best understood by imagining each symbol to be prefixed according to the context in which it is used. The axioms (37)–(38) could thus be interpreted as follows:

$$\mathsf{ClassAssertion}(\mathit{Cls\text{-}Eagle}\ \mathit{Ind\text{-}Harry}) \tag{39}$$
$$\mathsf{ClassAssertion}(\mathit{Cls\text{-}Endangered}\ \mathit{Ind\text{-}Eagle}) \tag{40}$$

Because the names of concepts and individuals do not interact even if they are the same, this type of metamodeling is often referred to as *punning*.

3.5. *Annotations of Axioms and Entities*

To address the limitations described in Section 2.5, in OWL 2 it is possible to annotate axioms as well as ontologies and entities. This can be used to capture additional information about axioms, such as their provenance or certainty values [34,8]. For example, the following axiom states that all humans are animals, and attributes that statement to Peter:

$$\begin{array}{l}\mathsf{SubClassOf}(\\ \quad \mathsf{Annotation}(\mathit{attributedTo}\ \text{“Peter”})\\ \quad \mathit{Human}\ \mathit{Animal})\end{array} \tag{41}$$

In contrast to OWL 1, annotations in OWL 2 carry no formal semantics—that is, they do not affect the set of consequences that can be derived from an ontology. Annotations are, however, accessible in the structural specification of OWL 2. Thus, applications are expected to provide access to annotations by nonlogical means, such as APIs.

Although annotations do not affect the semantics of an ontology, they do affect their structural equivalence. Thus, for two axioms to be structurally equivalent, the annotations on them must be structurally

equivalent as well, and similarly for ontologies. For example, the axiom above is semantically equivalent but not structurally equivalent to the following one:

$$\mathsf{SubClassOf}(\mathit{Human}\ \mathit{Animal}) \tag{42}$$

3.6. *Ontology Publishing, Imports, and Versioning*

In order to address the problems described in Section 2.4, OWL 2 precisely specifies the way in which ontologies are to be published on the Web. In particular, each ontology $\mathcal{O}$ can be given an ontology URI *ou* that uniquely identifies the ontology; if this is the case, then $\mathcal{O}$ should be located at location *ou*. Locating the ontologies imported into $\mathcal{O}$ is thus straightforward in OWL 2: the ontology $\mathcal{O}$ always specifies the location of the imported ontology. Thus, in essence, the import mechanism of OWL 2 is "by name and location."

The main difference to OWL 1 is that OWL 2 allows implementations to provide a suitable *redirection* mechanism: when requested to access an ontology from a location u, a tool can redirect u to a different location v and retrieve the ontology from there. The accessed ontology should, however, be treated just as though it had been retrieved from u. For example, relative URIs in the ontology should be resolved as if the ontology had been retrieved from u, and the ontology URI, if present, should be equal to u. The design of concrete redirection mechanisms is left entirely to OWL 2 implementations.

OWL 2 also provides a simple method for the management of ontology versions. In addition to an ontology URI, an OWL 2 ontology can contain a version URI, which identifies the version of the ontology. In such cases, the ontology URI identifies an *ontology series*—a set of all versions of a particular ontology—whereas the version URI identifies a particular version in the series.

The coexistence of different versions is enabled by relaxing the publishing restrictions mentioned earlier in this section: if an ontology $\mathcal{O}$ has an ontology URI *ou* and a version URI *vu*, then $\mathcal{O}$ should be located at *vu*; furthermore, if $\mathcal{O}$ is the current version of the ontology series, then it should be located at both *ou* and *vu*. Thus, a particular version of an ontology from a series can always be accessed from the location identified by the version URI, and the current version from the series can be accessed at the location identified by the ontology URI. (As usual, all these locations can be redirected in a way that is transparent to the ontologies in question.) Publishing a new current version of an ontology thus involves placing the new ontology in the appropriate location as identified by the version URI, and replacing the ontology located at the ontology URI with the new ontology.

3.7. *Functional-Style Syntax*

In addition to the MOF metamodel, the OWL 2 specification defines a Functional-Style Syntax as a simple encoding of the metamodel. The OWL 2 Functional-Style Syntax and the OWL 1 Abstract Syntax differ in many respects.

The main difference with respect to the OWL 1 Abstract Syntax is that the Functional-Style Syntax of OWL 2 does not contain the frame-like syntactic constructs of OWL 1; instead, ontology entities are described at a more fine-grained level using axioms. As explained in Section 2.2.1, the frame-like constructs in OWL 1 caused a significant amount of confusion in the OWL developer community. The new Functional-Style Syntax is more verbose than the old Abstract Syntax; however, this is generally not considered to be a problem: most ontologies are created using editors such as Protégé, which are free to present the information to the user in a compact, or even frame-like way.

The OWL 2 Functional-Style Syntax is *not* backwards compatible with the OWL 1 Abstract Syntax: an ontology written in OWL 1 Abstract Syntax may not be a valid OWL 2 ontology. The Working Group did not, however, consider this lack of backwards compatibility to be a problem in practice: the OWL 1 Abstract Syntax was used mainly for defining the structure and semantics of OWL 1, and has rarely been used to publish ontologies on the Web.

3.8. *XML Syntax*

As mentioned in Section 2.2.4, OWL 1 RDF is the syntax that has been mainly been used for publishing OWL 1 ontologies on the Web. This syntax, however, suffers from a number of problems. In order to provide an alternative to RDF, OWL 2 comes with an XML Syntax which presents a number of advantages for publishing ontologies on the Web. In particular, the new XML Syntax is easy to parse and process; also, XML is a widely adopted format on the Web and it is supported by a variety of tools.

The XML Syntax is also well-suited for use in protocols and APIs for accessing OWL 2 implementa-

tions. The Description Logics Implementors Group (DIG) [17] interface specification has established itself in the DL community as a de-facto standard for accessing DL reasoners. It is supported by virtually all DL reasoners and many ontology editors. DIG is based on exchanging XML messages over HTTP. In its previous incarnations, it used a bespoke XML language for describing ontologies, which was not completely aligned with OWL 1. In its newest incarnation (version 2.0), DIG reuses the ontology language and the syntax of OWL 2, thus reducing the implementation burden on developers. The XML Syntax for OWL 2 is obtained from the MOF metamodel by a straightforward translation of the MOF diagrams into XML Schema.

3.9. *RDF Serialization*

The vast majority of OWL 1 ontologies have been published online using the OWL 1 RDF syntax. For OWL 2 to be successful, it is important to ensure that existing OWL 1 ontologies can still be used and further developed with OWL 2 implementations. Therefore, an important requirement in the development of OWL 2 was to maintain backwards compatibility with the RDF syntax of OWL 1. At the same time, a major design goal in OWL 2 was to clean up problems in the definition of OWL 1 as discussed in Section 2.

This dilemma was resolved in OWL 2 via the following compromise. On the one hand, the conceptual structure of OWL 1 was reengineered without considering backwards compatibility, and the Functional-Style Syntax and the XML Syntax were changed accordingly. On the other hand, these changes were not reflected in the OWL 2 RDF syntax; rather, the OWL 2 RDF syntax has been designed so that every OWL 1 DL ontology written in RDF is also a valid OWL 2 ontology. The two representations are connected by a bidirectional translation between the Functional-Style Syntax (or, equivalently, the MOF metamodel) and RDF graphs. In contrast to OWL 1, the OWL 2 specification contains an explicit mapping from RDF graphs into the Functional-Style Syntax, which solves the ambiguity problems mentioned in Section 2.2.4.

The translations have been designed such that every OWL 2 ontology in the Functional-Style Syntax can be mapped into RDF triples and back without any change in the meaning of the ontology. More precisely, suppose that $\mathsf{RDF}(\mathcal{O})$ is the set of triples obtained by serializing the OWL 2 ontology $\mathcal{O}$ into RDF and assume that $\mathcal{O}'$ is the OWL 2 ontology obtained by applying the reverse transformation to $\mathsf{RDF}(\mathcal{O})$. Then, $\mathcal{O}$ and $\mathcal{O}'$ are logically equivalent (i.e., they have exactly the same models).

The ontologies $\mathcal{O}$ and $\mathcal{O}'$, even if logically equivalent, can still be structurally different; however, the mappings have been designed such that the structure of the ontologies is preserved whenever possible. Thus, syntactic changes may arise only if $\mathcal{O}$ uses certain OWL 2 features, which we briefly discuss in the rest of this section; in all other cases, the structure of ontologies is preserved.

First, since representing n-ary associations in RDF is cumbersome, certain axioms that involve more than two objects are translated into several independent axioms involving only two objects. For example, axiom

$$\mathsf{EquivalentClasses}(A\ B\ C\ D) \tag{43}$$

is serialized into RDF as the following triples:

$$\langle A, \mathit{owl{:}equivalentClass}, B\rangle \tag{44}$$
$$\langle B, \mathit{owl{:}equivalentClass}, C\rangle \tag{45}$$
$$\langle C, \mathit{owl{:}equivalentClass}, D\rangle \tag{46}$$

On these triples, the reverse mapping produces the following axioms:

$$\mathsf{EquivalentClasses}(A\ B) \tag{47}$$
$$\mathsf{EquivalentClasses}(B\ C) \tag{48}$$
$$\mathsf{EquivalentClasses}(C\ D) \tag{49}$$

Clearly, (43) and (47)–(49) are logically, but not structurally equivalent.

Second, since RDF does not allow blank nodes to occur in the property position, inverse properties in property assertions are translated into equivalent assertions without the inverse property. For example, axiom

$$\mathsf{PropertyAssertion}(\mathsf{InverseOf}(P)\ o_1\ o_2) \tag{50}$$

is translated into the following triple:

$$\langle o_2, P, o_1\rangle \tag{51}$$

The reverse mapping then produces the following axiom, which is semantically equivalent but not structurally equivalent with the original one:

$$\mathsf{PropertyAssertion}(P\ o_2\ o_1) \tag{52}$$

Third, annotations containing punned entities may not be correctly preserved. For example, assume that an ontology $\mathcal{O}$ uses a URI u both as

[17] `http://dig.cs.manchester.ac.uk/`

a class and an individual, but it defines different annotations for these two usages of U:

$$\mathsf{EntityAnnotation}(\ \mathsf{Class}(u)\ \mathsf{Annotation}(P\ a_1)) \tag{53}$$

$$\mathsf{EntityAnnotation}(\ \mathsf{NamedIndividual}(u)\ \mathsf{Annotation}(Q\ a_2)) \tag{54}$$

The translation of $\mathcal{O}$ into RDF contains the following triples:

$$\langle u, P, a_1 \rangle \tag{55}$$

$$\langle u, Q, a_2 \rangle \tag{56}$$

When translating these triples back into the structural specification, it is not possible to reproduce the original structure since the information that a_1 has been attached to the usage of u as a class and a_2 to the usage of u as an individual has been lost.

Finally, the RDF syntax of OWL 2 is backwards-compatible with that of OWL 1 DL in the sense that every OWL 1 DL ontology $\mathcal{O}_1$ in RDF syntax can be mapped into a valid OWL 2 ontology $\mathcal{O}_2$ using the reverse transformation from RDF to OWL 2 such that $\mathcal{O}_2$ has exactly the same set of models as $\mathcal{O}_1$, except for the effects due to annotations. As mentioned in Section 2.5, annotation properties are interpreted as binary predicates in OWL 1; in OWL 2, however, they are not given any semantics. Even so, the semantics of the rest of the ontology is preserved: every model of $\mathcal{O}_1$ is a model of $\mathcal{O}_2$ and any model of $\mathcal{O}_2$ can be extended to a model of $\mathcal{O}_1$ by interpreting the annotation properties.

3.10. *Semantics*

In order to avoid the problems described in Section 2.6, OWL 2 defines a clean direct model-theoretic semantics that clearly corresponds to the description logic $\mathcal{SROIQ}(\mathbf{D})$—a language obtained by extending $\mathcal{SROIQ}$ with datatypes. It has been defined for ontologies in Functional-Style Syntax; however, since this syntax is in one-to-one correspondence with the MOF metamodel and the XML Syntax, the semantics can directly be applied to these representations as well. The clear correspondence between $\mathcal{SROIQ}(\mathbf{D})$ and OWL 2 facilitates the direct reuse of the reasoning algorithms available for $\mathcal{SROIQ}(\mathbf{D})$ and therefore support for OWL 2 in existing DL reasoners.

At the moment, OWL 2 does not provide an RDF-style semantics. Therefore, in OWL 2, the transformation of ontologies in Functional-Style Syntax into RDF graphs is a purely syntactic process: the triples obtained by the RDF serialization are not intended to be interpreted directly, and the meaning of the RDF semantics on them does not define the meaning of the corresponding OWL 2 ontology.

There is, however, an interest within the Working Group in providing an RDF-style semantics to OWL 2. Currently, the design of such a semantics is work in progress. We believe, however, that the requirement for full semantic compatibility with RDF is of doubtful benefit: to the best of our knowledge, the systems that rely on the RDF-compatible semantics of OWL 1 do not fully support it. Moreover, the definition of an RDF semantics for the whole language will exacerbate the problems described in Section 2.6 and it may prevent further extensions to OWL 2 since the RDF semantics is not robust under extensions.

3.11. *Profiles*

To address the problems outlined in Sections 2.8 and 2.9, the OWL 2 specification identifies several profiles—trimmed down versions of the language that trade some expressive power for efficiency of reasoning. In logic, profiles are usually called fragments or sublanguages. OWL 2 provides three profiles—OWL 2 EL, OWL 2 QL, and OWL 2 RL—each of which provides different expressive power and targets different application scenarios.

The OWL 2 profiles are defined by placing restrictions on the Functional-Style Syntax of OWL 2. An ontology written in any of these profiles is a valid OWL 2 ontology. Therefore, the semantics of the OWL 2 profiles is given by the direct model theoretic semantics of OWL 2. Ontology modelers who want to ensure that their ontologies are in a certain profile can use these restrictions as a guide; furthermore, tool developers can easily use the corresponding grammars to create tools for checking which profile an ontology belongs to.

Apart from the profiles specified here, there are many other possible profiles of OWL 2. For example, one could define a subset of the OWL 2 Functional-Style syntax that corresponds to OWL 1 Lite or OWL 1 DL. Therefore, those subsets could also be viewed as OWL 2 profiles. We next discuss the features and applicability of each of the profiles of OWL 2.

3.11.1. *The OWL 2 EL Profile*

OWL 2 EL is based on the $\mathcal{EL}$++ family of description logics, which have been designed to allow for efficient reasoning with large terminologies [1]. The main reasoning service of interest is *classification*—the computation of the subclass relation between all the classes in an ontology. Reasoning in this profile can be implemented in time that is polynomial in the size of the ontology [1].

The central modeling features of this profile are class conjunction and SomeValuesFrom restrictions. In order to achieve tractability, the use of negation, disjunction, AllValuesFrom restrictions, and cardinality restrictions is disallowed. Many large-scale ontologies can be captured using this profile; a prominent example is the medical ontology SNOMED CT. In particular, OWL 2 EL captures a very common pattern used in ontologies for defining concepts, namely the combined use of conjunction and existential quantification. For example, it allows one to state that each heart contains a left ventricle and a right ventricle as its parts.

3.11.2. *The OWL 2 QL Profile*

OWL 2 QL is based on the DL-Lite family of description logics [7], which has been designed to allow for efficient reasoning with large amounts of data structured according to relatively simple schemata. The main reasoning service in the this profile is conjunctive query answering: given an ontology $\mathcal{O}$ and a conjunctive query q, the problem is to compute all tuples of individuals that constitute an answer to q w.r.t. $\mathcal{O}$. This task can be implemented by first rewriting q into a union of conjunctive queries uq that captures the information implicit in $\mathcal{O}$; then, uq can be answered over the data set by using conventional relational database techniques.

The profile provides most features needed to capture conceptual models, such as UML class diagrams, ER diagrams, and database schemas. For example, it provides means for representing class disjointness, the domain and range of properties, and participation constraints (e.g., every paper has an author). In order to ensure that each ontology can indeed be rewritten into a union of conjunctive queries, however, OWL 2 QL forbids the use of disjunction and AllValuesFrom restrictions, as well as certain other features that require recursive query evaluation.

3.11.3. *The OWL 2 RL Profile*

OWL 2 RL has been designed such that several reasoning tasks can be implemented as a set of rules in a forward-chaining rule system. To achieve this implementability criterion, OWL 2 RL is not as expressive as OWL 2. OWL 2 RL is attractive in situations where a limited extension of RDF-Schema is desired.

OWL 2 RL allows for most constructs of OWL 2; however, to allow for rule-based implementations of reasoning, the way these constructs can be used in axioms has been restricted. These restrictions ensure that a reasoning engine only needs to reason with the individuals that occur explicitly in the ontology. Thus, in OWL 2 RL it is not possible to use SomeValuesFrom restrictions on the right-hand side of a subclass axiom, as this would imply the existence of an "anonymous" individual—that is, an individual that is not explicitly known by name. Similar principles have been followed in the design of Description Logic Programs (DLP) [9].

The OWL 2 RL specification also provides a set of first-order implications that can directly be applied to an RDF graph in order to derive the relevant consequences. These implications are reminiscent of the pD^* semantics for OWL 1 [35], and they provide a useful starting point for the implementation of forward-chaining reasoners for OWL 2 RL. These first-order implications are equivalent to the direct model-theoretic semantics of OWL 2 in a certain well-defined sense: if $\mathcal{O}$ is an OWL 2 RL ontology in Functional-Style Syntax and it does not employ punning, then $\mathcal{O}$ and its RDF encoding $\mathsf{RDF}(\mathcal{O})$ entail the same set of ground facts.

4. Implementation and Adoption

Considerable progress has been achieved in the development of tool support for OWL 2. This is due to the improvements made in the specification which have facilitated the implementors' work, as well as to the pressing need in practice for the additional functionality provided by OWL 2. The development activity can be grouped into two major categories: the extension of existing APIs and ontology management systems to the new syntax and the MOF metamodel, and the extension of reasoning systems to support the new expressive features.

The new syntax is currently supported by the new version of the OWL API. The widely used Protégé system has recently been extended with support for

the additional constructs provided by OWL 2 [14]. The commercial tool TopBraid composer also currently supports OWL 2. The KAON2 API is in the process of being extended to make it compliant with the MOF metamodel of OWL 2, and we hope that other widely used APIs will soon follow suit.

Support for OWL 2 has also been included into the FaCT++ and the Pellet systems. The developers of these tools have not reported any difficulties in extending their existing implementations with the new constructs, and it is hoped that other major DL reasoning systems, such as RACER and HermiT, will soon follow suit and support $\mathcal{SROIQ}$.

Furthermore, there are already stable implementations of the OWL 2 profiles. OWL 2 EL has been implemented in the CEL system; [18] Arity Corporation has also made a commercial implementation available. [19] OWL 2 QL has been implemented in the QuONto [20] and Owlgres [21] systems. The OWL 2 RL profile has been recently implemented in Oracle's database system version 11g.

Concerning the adoption of the language, OWL 2 has already been used in several application areas, such as bioinformatics [33,38], clinical sciences [10], engineering [15], and law [13]. This is partly due to the availability of editors and reasoners for OWL 2. We expect that, as the tools continue to improve, OWL users will increasingly migrate to OWL 2 and will start using the new language features. Moreover, we expect that the specification of profiles and the availability of reasoning engines that allow for tractable forms of reasoning will facilitate the use of OWL by users who were initially discouraged by scalability and efficiency issues.

5. The Future of OWL

Apart from addressing acute problems with expressivity, a goal in the development of OWL 2 was to provide a robust platform for future development. We conclude this paper by speculating about potential future extensions.

Syntactic Extensibility. Since the first OWL-ED workshop, many proposals for syntactic sugar were put forward. In particular, extending OWL with a "macro" system, allowing users to define their syntactic shortcuts, might be very useful. For example, one could define SomeAllValuesFrom(P A) as a syntactic shortcut for a SomeValuesFrom restriction conjoined with an AllValuesFrom.

Query Languages. Expressive query languages are very important for practical applications of OWL. Variants of conjunctive queries have repeatedly been proposed, and were extended to algebraic languages in proposals such as EQL-Lite [6] and nRQL [39]. The challenge for the OWL development community is to clarify the exact relationships between these proposals, as well as the relationship with the RDF query language SPARQL.

Integration with Rules. The W3C Rules Interchange Format (RIF) Working Group [22] is currently working on producing an interchange format for the rules on the Web. Furthermore, many approaches to integrating rules with DL-based languages were proposed in the past couple of years. Hence, integrating the RIF effort with the development of OWL is a task of critical importance to both communities. A set of rules that are of particular relevance are the so-called DL-safe rules [25] since they provide useful expressive power while still allowing for practically feasible implementations.

Nonmonotonic Extensions. Users have repeatedly asked for nonmonotonic extensions of OWL, as they are crucial to solving advanced modeling problems such as exceptions or database-like constraints.

6. Acknowledgements

Major groups involved in the development of OWL 2 include the the World Wide Web Consortium and the OWL Working Group. There are far too many people involved to list them individually, even if limited to the major participants. The views on the design decisions of OWL expressed in this paper are those of the authors, and do not necessarily reflect those of other members of the OWL Working Group. Finally, by the time this paper was completed, the Working Group was still active, so some of the information given here may become obsolete in the future. The final version of OWL 2, however, is unlikely to depart significantly from the description given in this paper. We have pointed out the precise aspects of the language that are still under discussion and may be subject to change.

[18] http://lat.inf.tu-dresden.de/systems/cel/
[19] http://www.arity.com/
[20] http://www.dis.uniroma1.it/~quonto
[21] http://pellet.owldl.org/owlgres/
[22] http://www.w3.org/2005/rules/

References

[1] F. Baader, S. Brandt, C. Lutz, Pushing the $\mathcal{EL}$ Envelope, in: Proc. of IJCAI 2005, 2005.

[2] F. Baader, P. Hanschke, A Scheme for Integrating Concrete Domains into Concept Languages, in: Proc. IJCAI '91, Sydney, Australia, 1991.

[3] F. Baader, U. Sattler, Description logics with aggregates and concrete domains, Information Systems 28 (8) (2003) 979–1004.

[4] S. Bechhofer, Parsing OWL in RDF/XML, W3C Working Group Note (January 21 2004).

[5] S. Bechhofer, J. Carroll, OWL DL: Trees or Triples?, in: Proc. WWW 2004, New York, USA, 2004.

[6] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Epistemic First-Order Queries over Description Logic Knowledge Bases, in: Proc. DL 2006, Lake District, UK, 2006.

[7] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family, Journal of Automated Reasoning 9 (2007) 385–429.

[8] T. T. Duc, P. Haase, B. Motik, B. C. Grau, I. Horrocks, Metalevel Information in Ontology-Based Applications, in: Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI 2008), Chicago, IL, USA, 2008, to appear.

[9] B. N. Grosof, I. Horrocks, R. Volz, S. Decker, Description Logic Programs: Combining Logic Programs with Description Logic, in: Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, 20-24 May, 2003.

[10] M. Gustafsson, G. Falkman., Experiences in Modeling Clinical Examinations in Oral Medicine Using OWL, in: owl-07, 2007.

[11] V. Haarslev, R. Möller, RACER System Description, in: Proc. IJCAR 2001, Siena, Italy, 2001.

[12] P. Hayes, RDF Model Theory, W3C Recommendation (10 February 2004).

[13] R. Hoekstra, Use of OWL in the Legal Domain, in: owl-08, 2008.

[14] M. Horridge, D. Tsarkov, Supporting Early Adoption of OWL 1.1 with Protege-OWL and FaCT++, in: Proc. OWL-ED 2006, Athens, GA, USA, 2006.

[15] I. Horrocks, H. Graves, Applications of OWL 1.1 to Systems Engineering, in: owl-08, 2008.

[16] I. Horrocks, O. Kutz, U. Sattler, The Even More Irresistible $\mathcal{SROIQ}$, in: Proc. KR 2006, Lake District, UK, 2006.

[17] I. Horrocks, P. F. Patel-Schneider, Reducing OWL Entailment to Description Logic Satisfiability, Journal of Web Semantics 1 (4) (2004) 345–357.

[18] I. Horrocks, P. F. Patel-Schneider, F. van Harmelen, From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language, Journal of Web Semantics 1 (1) (2003) 7–26.

[19] I. Horrocks, U. Sattler, A Tableaux Decision Procedure for $\mathcal{SHOIQ}$, in: Proc. of IJCAI 2005, 2005.

[20] I. Horrocks, U. Sattler, S. Tobies, Practical Reasoning for Very Expressive Description Logics, Logic Journal of the IGPL 8 (3) (2000) 239–263.

[21] A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca-Grau, J. Hendler, SWOOP: a Web Ontology Editing Browser, Journal of Web Semantics 4 (2) (2005) 144–153.

[22] Y. Kazakov, $\mathcal{SRIQ}$ and $\mathcal{SROIQ}$ are Harder than $\mathcal{SHOIQ}$, in: Proc. of DL 2008, 2008.

[23] C. Lutz, C. Areces, I. Horrocks, U. Sattler, Keys, Nominals, and Concrete Domains, J. Artif. Intell. Res. (JAIR) 23 (2005) 667–726.

[24] B. Motik, On the Properties of Metamodeling in OWL, in: Proc. of ISWC 2005, 2005.

[25] B. Motik, U. Sattler, R. Studer, Query Answering in OWL-DL with Rules, in: Proc. of the Third International Semantic Web Conference (ISWC 2004), 2004.

[26] B. Motik, R. Shearer, I. Horrocks, Optimized Reasoning in Description Logics Using Hypertableaux, in: CADE-07, 2007.

[27] P. Patel-Schneider, Building the Semantic Web Tower from RDF Straw, in: Proc. of IJCAI 2005, 2005.

[28] P. F. Patel-Schneider, P. Hayes, I. Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation (10 February 2004).

[29] A. Rector, Analysis of Propagation along Transitive Roles: Formalisation of the Galen Experience with Medical Ontologies, in: Proc. DL 2002, Toulouse, France, 2002.

[30] A. Rector, G. Schreiber, Qualified Cardinality Restrictions (QCRs): Constraining the Number of Values of a Particular Type for a Property, W3C Working Draft (November 2 2005).

[31] A. Rector, C. Welty, Simple Part-whole Relations in OWL Ontologies, W3C Working Draft (August 11 2005).

[32] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, J. Web Sem. 5 (2) (2007) 51–53.

[33] R. Stevens, M. E. Aranguren, K. Wolstencroft, U. Sattler, N. Drummond, M. Horridge, A. Rector, Using OWL to Model Biological Knowledge, International Journal of Human Computer Studies 65 (2007) 583594.

[34] G. Stoilos, G. Stamou, V. Tzouvaras, J. Z. Pan, I. Horrocks, Fuzzy OWL: Uncertainty and the Semantic Web, in: OWL: Experiences and Directions (OWL-ED 2005), 2005.

[35] H. J. ter Horst, Completeness, Decidability and Complexity of Entailment for RDF Schema and a Semantic Extension Involving the OWL Vocabulary, J. Web Semantics 3 (2-3) (2005) 79–115.

[36] S. Tobies, Complexity Results and Practical Algorithms for Logics in Knowledge Representation, Ph.D. thesis, RWTH Aachen (2001).

[37] D. Tsarkov, I. Horrocks, FaCT++ Description Logic Reasoner: System Description, in: Proc. IJCAR 2006, Seattle, WA, USA, 2006.

[38] N. Villanueva-Rosales, M. Dumontier, Modeling Life Science Knowledge with OWL 1.1, in: owl-08, 2008.

[39] M. Wessel, R. Möller, A High Performance Semantic Web Query Answering Engine, in: Proc. DL 2005, Edinburgh, UK, 2005.

[40] K. Wolstencroft, A. Brass, I. Horrocks, P. W. Lord, U. Sattler, D. Turi, R. Stevens, A Little Semantic Web Goes a Long Way in Biology, in: ISWC-05, 2005.