14

Digital Libraries and Web-Based Information Systems

Ian Horrocks

Deborah L. McGuinness

Christopher A. Welty

Abstract

It has long been realised that the web could benefit from having its content understandable and available in a machine processable form, and it is widely agreed that ontologies will play a key role in providing much enabling infrastructure to support this goal. In this chapter we review briefly a selected history of description logics in web-based information systems, and the more recent developments related to OIL, DAML+OIL and the semantic web. OIL and DAML+OIL are ontology languages specifically designed for use on the web; they exploit existing web standards (XML, RDF and RDFS), adding the formal rigor of a description logic and the ontological primitives of object oriented and frame based systems.

14.1 Background and history

The research world as well as the general public are unified in their agreement that the web would benefit from some structure and explicit semantics for at least some of its content. Numerous companies exist today whose entire business model is based on providing some semblance of structure and conceptual search (i.e., yellow pages and search).

To paraphrase Milne [1928], "Providing structure is one of the things description logics do best!". In this chapter we review briefly the history of description logics in web-based information systems, and the more recent developments related to OIL (the Ontology Inference Layer), DAML (the DARPA Agent Markup Language), DAML+OIL and the "semantic web."

The web has been a compelling place for research activity in the last few years, and as we can not cover all the many efforts we will choose a few exemplar efforts that illustrate some of the key issues related to description logics on the web.

14.1.1 Untangle

The relationship between hypertext and semantic networks has long been realized, but one of the earliest description logic systems to realize this relationship was the UNTANGLE system [Welty and Jenkins, 2000], a description-logic system for representing bibliographic (card-catalog) information. The UNTANGLE Project began as a bit of exploratory research in using description logics for digital libraries [Welty, 1994], but out of sheer temporal coincidence with the rise of the web, a web interface was added and the first web-based description logic system was born.

The original UNTANGLE web interface was developed in 1994 [Welty, 1996a], and combined LISP-CLASSIC and the COMMONLISP Hypermedia Server (CL-HTTP) [Mallery, 1994] to implement a hypertext view of the ABox and TBox semantic networks, and used nested bullet lists to view the concept taxonomy, with in-page cross references for concepts having multiple parents. The interface was interesting in some respects as a tool to visualize description logic and semantic network information, though this aspect was never fully developed.

The research in the UNTANGLE project was to apply description logics to problems in digital libraries, specifically the classification and retrieval of card catalog information. In the early days of description logic applications, researchers scoured the world for taxonomies. One place with well-developed taxonomies are library subject classifications schemes, such as the Dewey Decimal System. The UNTANGLE project sought to utilize description logics to formally represent the established and well-documented processes by which books are classified by subject, with the goal of providing a tool to improve accuracy and increase the throughput of classification. The promise of digital libraries clearly seemed to imply that the entirely human-based system of subject classification would become backlogged and a hindrance to publication.

While the main contribution of the work was actually in the area of digital library ontologies, it had several useful implications for description logics. For conceptual modeling, the system made clear the very practical uses for primitive and defined concepts as basic ontological notions. Primitive concepts can be used in a model to represent classes of individuals that users are expected to be able to classify naturally. Defined concepts can be used in a model to represent subclasses of the primitive ones that the system will be able to classify if needed. For example, in libraries we expect a librarian to be responsible for recognizing the difference between a book and a journal. Such a distinction is trivial. On the other hand, they are not responsible for classifying a biography (though they can, of course): a biography is simply a book whose subject is a person.

As the World Wide Web (WWW) became the primary means of dissemination of computer science research, the goals of the UNTANGLE project shifted in 1995 to cataloging and classifying pages on the web [Welty, 1996b], which was viewed as a massive and unstructured digital library [Welty, 1998]. A similar project began at roughly that time at AT&T, whose goal was to utilize CLASSIC to represent a taxonomy of web bookmarks. While never published, this early work by Tom Kirk was part of the information manifold project [Levy *et al.*, 1995]. Kirk's visualisation tools were also used internally to provide additional visualisation support to the CLASSIC system.

This new work exposed some of the limitations of using description logics for modeling [Welty, 1998]. One must trade-off utilizing automated support for subsumption with the need to reify the concepts themselves. For example, the work started with the motivation that library classification schemes were well-developed taxonomies that would be appropriate for use in description logics. To utilize the power of subsumption reasoning, the elements of the subject taxonomy must obviously be *concepts*. Some subjects, however, are also useful to consider as *individuals*. For example, Ernest Hemingway is a person, an author of several books, and therefore best represented as an individual. Hemingway is also, however, the subject of his (many) biographies, and therefore he must be represented as a concept in the subject taxonomy. This is a simple example of precisely the kind of representation that is difficult for a description logic, without inventing some special purpose "hack". Similar notions have also been reported in the knowledge engineering community [Wielinga *et al.*, 2001].

14.1.2 FindUR

Another early project using description logics for the web was the FINDUR system at AT&T. FINDUR [McGuinness, 1998; McGuinness *et al.*, 1997] was an excellent example of picking "low hanging fruit" for description logic applications. The basic notion of FINDUR was *query expansion*,[1] that is, taking synonyms or hyponyms (more specific terms) and including them in the input terms, thereby expanding the query.

Information retrieval, especially as it is available on the web, rates itself by two independent criteria, *precision* and *recall*. Precision refers to the ratio of desired to undesired pages returned by a search, and recall refers to the ratio of desired pages missed to the total number of desired pages. Alternate terms for these notions are false-positives and false-negatives.

One of the main causes of false negatives in statistically-based keyword searches

[1] Sometimes other correlated terms are also used in query expansion. In a later piece of work [Rousset, 1999b], similar because it considered a description logic-based approach for query expansion, more of the formal issues are addressed in evaluating the soundness and completeness of a particular approach. There have also been others who have considered description-logic approaches (or dl-inspired approaches) to retrieval, for example [Meghini *et al.*, 1997].

is the use of synonymous or hyponymous search terms. For example, on the (then) AT&T Bell Labs research site, short project descriptions existed about description logics. These never referred to the phrase "artificial intelligence". Thus, a search for the general topic "artificial intelligence" would miss the description logic project pages even though description logics is a sub-area of artificial intelligence. If the page referred to "AI" instead of "artificial intelligence" precisely, a keyword search would also miss this clear reference to the same thing. This is a well recognized failure of shallow surface search techniques that significantly impacts recall.

The FindUR system represented a simple background knowledge base containing mostly thesaurus information built in a description logic (Classic) using the most basic notions of Wordnet (synsets and hyper/hyponyms) [Miller, 1995]. Concepts corresponding to sets of synonyms (synsets) were arranged in a taxonomy. These synsets also contained an informal list of related terms. Site specific search engines (built on Verity—a commercial search engine) were hooked up to the knowledge base. Any search term would first be checked in the knowledge base, and if contained in any synset, a new query would be constructed consisting of the disjunction of all the synonymous terms, as well as all the more specific terms (hyponyms).

The background knowledge was represented in Classic, however the description logic was not itself part of the on-line system. Instead, the information used by the search engine was statically generated on a regular basis and used to populate the search engine. The true power of using a description logic as the underlying substrate for the knowledge base was realized mainly in the maintenance task. The DL allowed the maintainer of the knowledge base to maintain some amount of consistency, such as discovering cycles in the taxonomy and disjoint synsets. These simple constraints proved effective tools for maintaining the knowledge since the knowledge itself was very simple.

The FindUR system was deployed on the web to support the AT&T research web site and a number of other application areas. Although the initial deployments were as very simple query expansion, some later deployments included more structure. For example, the FindUR applications on newspaper sites and calendar applications (such as the Summit calendar[1]) included searches that could specify a date range, date ordered returns, and a few other search areas including region or topic area. These searches included use of metatagging information on dates, location, topics, sometimes author, etc. This functioned as a structured search similar in nature to the later developed SHOE Search [Heflin and Hendler, 2001] for the semantic web, and was also similar to what Forrester reported as being required for search that would support eCommerce [Hagen *et al.*, 1999]. The FindUR applications for medical information retrieval [McGuinness, 1999] also included more sophisticated

[1] `http://www.quintillion.com/summit/calendar/`

mechanisms that allowed users to search in order of quality of study method used (such as randomized control trial study). Applications of FINDUR ranged in the end to include very simple query expansion, such as those deployed on WorldNet and Quintillion (see Directory Westfield[2]), as well as more complicated markup search such as those on the AT&T competitive intelligence site and the P-CHIP primary care literature search.

14.1.3 From SGML to the Semantic Web

Independent of description logics, and dating back to the mid 1980s, researchers in other areas of digital libraries were using SGML[1] (Standard Generalized Markup Language) as a tool to mark up a variety of elements of electronic texts, such as identifying the characters in novels, cities, etc., in order to differentiate them in search. For example, a reference to Washington the person in some text may appear as `<person>Washington</person>` whereas a reference to the U.S. State may be `<state>Washington</state>`. See, for example, the 1986 Text Encoding Initiative [Mylonas and Renear, 1999]. Clearly, a search tool capable of recognizing these tags would be more precise when searching for "Washington the person". This work may be viewed as establishing some of the ground work for the vision of the semantic-web that Tim Berners-Lee and colleagues have more recently popularized.

As the SGML communities proceeded in their efforts to create large repositories of "semantically" marked-up electronic documents, research in using these growing resources sprang up the database and description logics communities, with some early results making it clear that description logics were powerful tools for handling semi-structured data [Calvanese *et al.*, 1998c; 1999d].

In the mid 1990s, work in SGML gained some attention mainly because HTML[2] (HyperText Markup Language) was an SGML technology, and it became clear that the same sort of "semantic" markup (as opposed to "rendering" markup) could be applied to web pages, with the same potential gains. The main syntax specification properties of SGML were combined with the text rendering properties of HTML to generate XML[3] (Extensible Markup Language), and with it came the promise of a new sort of web, a web in which "meta data" would become the primary consumer of bandwidth. These connections made it reasonable to consider the existing work on semi-structured data in description logics a web application.

In an attempt to prevent the web community from repeating the same mistakes made in knowledge representation in the 1970s, in particular using informal "picture" systems with no understood semantics and without decidable reasoning, the

[2] `http://www.ataclick.com/westfield/`
[1] `http://www.w3.org/MarkUp/SGML/`
[2] `http://www.w3.org/MarkUp/`
[3] `http://www.w3.org/XML/`

description logics community became very active in offering languages for the new semantic web. The community was already well-positioned to influence the future of semantic web standards, due in part to (a) the strong history that description logics bring, with well researched and articulated languages providing clear semantics (as well as complexity analyses), (b) the existing work on the web described here, including web applications like UNTANGLE and FINDUR, and (c) description logic languages designed for web use such as OIL.

14.2 Enabling the Semantic Web: DAML

The web, while wildly successful in growth, may be viewed as being limited by its reliance on languages like HTML that are focused on *presentation* of information (i.e., text formatting). Languages such as XML do add some support for capturing the meaning of terms (instead of simply how to render a term in a browser), however it is widely perceived that more is needed. The DARPA Agent Markup Language (DAML) program[1] was one of the programs initiated in order to provide the foundation for the next generation of the web which, it is anticipated, will increasingly utilize agents and programs rather than relying so heavily on human interpretation of web information [Hendler and McGuinness, 2000]. In order for this evolution to occur, agents and programs must understand how to interact with information and services available on the web. They must understand what the information means that they are manipulating and also must understand what services can be provided from applications. Thus, meaning of information and services must be captured. Languages and environments existing today are making a start at providing the required infrastructure. The DAML program exists in order to provide funding for research on languages, tools, and techniques for making the web machine understandable.

The groundwork for the DAML program was being laid in 1999 with the approval for the broad area announcement in November and a web semantics language workshop in December 1999. A strawman language proposal effort was begun out of that work and the major initial emphasis began with a web-centric view. A web-oriented strawman proposal was worked on but not widely announced. One of the early widely-distributed contributions of the DAML program was DAML-ONT [2]—a proposal for an ontology language for the web [Hendler and McGuinness, 2000; McGuinness *et al.*, 2002]. This language began with the requirement to build on the best practice in web languages of the time and took the strawman proposal as the motivating starting point. That meant beginning with XML, RDF [3] (Re-

[1] `http://www.daml.org/`
[2] `http://www.daml.org/2000/10/daml-ont.html`
[3] `http://www.w3.org/RDF/`

source Description Framework), and RDFS [4] (RDF Schema). These languages were not expressive enough to capture the meaning required to support machine understandability, however, so one requirement was additional expressive power. The goal in choosing the language elements was to include the commonly used modeling primitives from object-oriented systems and frame-based systems. Finally, the community recognized the importance of a strong formal foundation for the language. Description logics as a field has had a long history of providing a formal foundation for a family of frame languages. Description logic languages add constructors into a language only after researchers specify and analyze the meaning of the terms and their computational effect on systems built to reason with them. The DAML community wanted to include the strong formal foundations of description logics in order to provide a web language that could be understood and extended.

The initial DAML web ontology language (DAML-ONT) was released publicly in October 2000. While the language design attempted to meet all of the design goals, beginning with the web-centric vision and later incorporating some description logic aspects, the decision was made that a timely release of the initial language was more critical than a timely integration of a description logic language with the web language. Thus the initial release focused more on the goals of web language compatibility and mainstream object-oriented and frame system constructor inclusion. Although some notions of description logic languages and systems were integrated, the major integration happened in the next language release (DAML+OIL).

Another important effort began at about the same time (in 1999) and produced a distributed language specification prior[1] to DAML-ONT called OIL. The aims of OIL's developers were similar to those of the DAML group, i.e., to provide a foundation for the next generation of the web. Their initial objective was to create a web ontology language that combined the formal rigor of a description logic with the ontological primitives of object oriented and frame based systems. Like DAML-ONT, OIL had an RDFS based syntax (as well as an XML syntax). However, the developers of OIL placed a stronger emphasis on formal foundations, and the language was explicitly designed so that its semantics could be specified via a mapping to the description logic $\mathcal{SHIQ}$ [Fensel *et al.*, 2001; Horrocks *et al.*, 1999].

It became obvious to both 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 merged language has a formal (model theoretic) seman-

[4] `http://www.w3.org/TR/2000/CR-rdf-schema-20000327/`

[1] Presentations of the language were made, for example, at the Dagstuhl Seminar on Semantics for the Web—see http://www.semanticweb.org/events/dagstuhl2000/.

tics that provides machine and human understandability, an axiomatization [Fikes and McGuinness, 2001] that provides machine operationalization with a specification of valid inference "rules" in the form of axioms, and a reconciliation of the language constructors from the two languages.

14.3 OIL and DAML+OIL

14.3.1 OIL

The OIL language is designed to combine frame-like modelling primitives with the increased (in some respects) expressive power, formal rigor and automated reasoning services of an expressive description logic [Fensel *et al.*, 2000]. OIL also comes "web enabled" by having both XML and RDFS based serialisations (as well as a formally specified "human readable" form, which we will use here). The frame structure of OIL is based on XOL [Karp *et al.*, 1999], an XML serialisation of the OKBC-lite knowledge model [Chaudhri *et al.*, 1998b]. In these languages classes (concepts) are described by *frames*, whose main components consist of a list of super-classes and a list of *slot-filler* pairs. A slot corresponds to a role in a DL, and a slot-filler pair corresponds to either a value restriction (a concept of the form $\forall R.C$) or an existential quantification (a concept of the form $\exists R.C$)—one of the criticisms leveled at frame languages is that they are often unclear as to exactly which of these is intended by a slot-filler pair.

OIL extends this basic frame syntax so that it can capture the full power of an expressive description logic. These extensions include:

- Arbitrary Boolean combinations of classes (called *class expressions*) can be formed, and used anywhere that a class name can be used. In particular, class expressions can be used as slot fillers, whereas in typical frame languages slot fillers are restricted to being class (or individual) names.
- A slot-filler pair (called a *slot constraint*) can itself be treated as a class: it can be used anywhere that a class name can be used, and can be combined with other classes in class expressions.
- Class definitions (frames) have an (optional) additional field that specifies whether the class definition is primitive (a subsumption axiom) or non-primitive (an equivalence axiom). If omitted, this defaults to primitive.
- Different types of slot constraint are provided, specifying value restriction, existential quantification and various kinds of cardinality constraint.[1]
- Global slot definitions are extended to allow the specification of superslots (subsuming slots) and of properties such as *transitive* and *symmetrical*.

[1] Some frame languages also provide this feature, referring to such slot constraints as *facets* [Chaudhri *et al.*, 1998b; Grosso *et al.*, 1999].

- Unlike many frame languages, there is no restriction on the ordering of class and slot definitions, so classes and slots can be used before they are "defined". This means that OIL ontologies can contain cycles.
- In addition to standard class definitions (frames), which can be seen as DL axioms of the form $CN \sqsubseteq C$ and $CN \equiv C$ where CN is a concept name, OIL also provides axioms for asserting disjointness, equivalence and coverings with respect to class expressions. This is equivalent to providing general inclusion (or equivalence) axioms, i.e., axioms of the form $C \sqsubseteq D$ ($C \equiv D$), where both C and D may be non-atomic concepts.

Many of these points are standard for a DL (i.e., treating $\forall R.C$ and $\exists R.C$ as classes), but are novel for a frame language.

OIL is also more restrictive than typical frame languages in some respects. In particular, it does not support collection types other than sets (e.g., lists or bags), and it does not support the specification of default fillers. These restrictions are necessary in order to maintain the formal properties of the language (e.g., monotonicity) and the correspondence with description logics (see Chapter 2).

In order to allow users to choose the expressive power appropriate to their application, and to allow for future extensions, a layered family of OIL languages has been described. The base layer, called "Core OIL" [Bechhofer *et al.*, 2000], is a cut down version of the language that closely corresponds with RDFS (i.e., it includes only class and slot inclusion axioms, and slot range and domain constraints[1]). The standard language, as described here, is called "Standard OIL", and when extended with ABox axioms (i.e., the ability to assert that individuals and tuples are, respectively, instances of classes and slots), is called "Instance OIL". Finally, "Heavy OIL" is the name given to a further layer that will include as yet unspecified language extensions.

We will only consider Standard OIL in this chapter: Core OIL is too weak to be of much interest, Heavy OIL has yet to be specified, and Instance OIL adds nothing but ABox axioms. Moreover, it is unclear if adding ABox axioms to OIL would be particularly useful as RDF already provides the means to assert relationships between (pairs of) web resources and the slots and classes defined in OIL ontologies.

Figure 14.1 illustrates an OIL ontology (using the human readable serialisation) corresponding to an example terminology from Chapter 2. The structure of the language will be described in detail in Section 14.3.1.1. A full specification of OIL, including DTDs for the XML and RDFS serialisations, can be found in [Horrocks *et al.*, 2000a] and on the OIL web site.[2]

[1] Constraining the range (respectively domain) of a slot SN to class C is equivalent to a DL axiom of the form $\top \sqsubseteq \forall SN.C$ (respectively $\exists SN.\top \sqsubseteq C$).

[2] `http://www.ontoknowledge.org/oil/`

```
name "Family"
documentation "Example ontology describing family relationships"
definitions
  slot-def hasChild
    inverse isChildOf
  class-def defined Woman
    subclass-of Person Female
  class-def defined Man
    subclass-of Person not Woman
  class-def defined Mother
    subclass-of Woman
    slot-constraint hasChild
      has-value Person
  class-def defined Father
    subclass-of Man
    slot-constraint hasChild
      has-value Person
  class-def defined Parent
    subclass-of or Father Mother
  class-def defined Grandmother
    subclass-of Mother
    slot-constraint hasChild
      has-value Parent
  class-def defined MotherWithManyChildren
    subclass-of Mother
    slot-constraint hasChild
      min-cardinality 3
  class-def defined MotherWithoutDaughter
    subclass-of Mother
    slot-constraint hasChild
      value-type not Woman
```

Fig. 14.1. OIL "family" ontology.

14.3.1.1 OIL syntax and semantics

OIL can be seen as a syntactic variant of the description logic $\mathcal{SHIQ}$ [Horrocks *et al.*, 1999] extended with simple concrete datatypes [Baader and Hanschke, 1991a; Horrocks and Sattler, 2001]; we will call this DL $\mathcal{SHIQ}(\mathcal{D})$. Rather than providing the usual model theoretic semantics, OIL defines a translation $\sigma(\cdot)$ that maps an OIL ontology into an equivalent $\mathcal{SHIQ}(\mathcal{D})$ terminology. From this mapping, OIL derives both a clear semantics and a means to exploit the reasoning services of DL systems such as FaCT [Horrocks, 1998b] and Racer [Haarslev and Möller, 2001e] that implement (most of) $\mathcal{SHIQ}(\mathcal{D})$.

The translation is quite straightforward and follows directly from the syntax and informal specification of OIL. The single exception is in the treatment of OIL's `one-of` constructor. This is *not* treated like the DL `one-of` constructor described in Chapter 2, but is mapped to a disjunction of specially introduced disjoint primitive concepts corresponding to the individual names in the `one-of` construct, i.e., individuals are treated as primitive concepts, and there is an implicit unique name assumption. This was a pragmatic decision based on the fact that reasoning with individuals in concept descriptions is known to be of very high complexity (for a DL as expressive as OIL), and is beyond the scope of any implemented DL system—in fact a practical algorithm for such a DL has yet to be described [Horrocks and Sattler, 2001]. This treatment of the `one-of` constructor is not without precedent in DL systems: a similar approach was taken in the CLASSIC system [Borgida and Patel-Schneider, 1994].

An OIL ontology consists of a *container* followed by a list of *definitions*. The container consists of Dublin Core compliant documentation fields specifying, e.g., the title and subject of the ontology. It is ignored by the translation, and wont be considered here. Definitions can be either class definitions, axioms, slot definitions or import statements, the latter simply specifying (by URI) other ontologies whose definitions should be teated as being lexically included in the current one. We will, therefore, treat an OIL ontology as a list $A_1, \ldots, A_n$, where each A_i is either a class definition, an axiom or a slot definition. This list of definitions/axioms is translated into a $\mathcal{SHIQ}(\mathcal{D})$ terminology $\mathcal{T}$ (a set of axioms) as follows:

$$\sigma(A_1, \ldots, A_n) = \{\sigma(A_1), \ldots, \sigma(A_1)\} \cup \bigcup_{1 \leqslant j < n} \bigcup_{j < k \leqslant n} \{P_j \sqsubseteq \neg P_k\}$$

where $i_1, \ldots, i_n$ are the individuals used in the ontology, and P_i is the $\mathcal{SHIQ}(\mathcal{D})$ primitive concept used to represent i.

Class definitions An OIL class definition (`class-def`) consists of an optional keyword K followed by a class name CN, an optional documentation string, and a class description D. If $K =$ `primitive`, or if K is omitted, then the class definition corresponds to a DL axiom of the form $CN \sqsubseteq D$. If $K =$ `defined`, then the class definition corresponds to a DL axiom of the form $CN \equiv D$.

A class description consists of an optional `subclass-of` component, with a list of one or more class expressions, followed by a list of zero or more `slot-constraints`. Each slot constraint can specify a list of constraints that apply to the given slot, e.g., value restrictions and existential quantifications. The set of class expressions and slot constraints is treated as an implicit conjunction.

The complete mapping from OIL class definitions to $\mathcal{SHIQ}(\mathcal{D})$ axioms is given in Figure 14.2, where CN is a class or concept name and C is a class expression.

OIL	$\mathcal{SHIQ}(\mathcal{D})$
`class-def` (`primitive` | `defined`) CN	CN ($\sqsubseteq$ | $\equiv$) $\top$
`subclass-of` $C_1 \ldots C_n$	$\sqcap\, \sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
slot-constraint$_1$	$\sqcap\, \sigma$(slot-constraint$_1$)
$\vdots$	$\vdots$
slot-constraint$_m$	$\sqcap\, \sigma$(slot-constraint$_m$)

Fig. 14.2. OIL to $\mathcal{SHIQ}(\mathcal{D})$ mapping (class definitions).

Slot constraints A `slot-constraint` consists of a slot name followed by a list of one or more constraints that apply to the slot. A constraint can be either:

- A `has-value` constraint with a list of one or more class-expressions or datatype expressions.
- A `value-type` constraint with a list of one or more class-expressions or datatype expressions.
- A `max-cardinality`, `min-cardinality` or `cardinality` constraint with a non-negative integer followed (optionally) by either a class expression or a datatype expression.
- A `has-filler` constraint with a list of one or more individual names or data values.

OIL `has-value` and `value-type` constraints correspond to DL existential quantifications and value restrictions respectively. OIL `cardinality` constraints correspond to DL qualified number restrictions, where the qualifying concept is taken to be $\top$ if the class expression is omitted. In order to maintain the decidability of the language, cardinality constraints can only be applied to *simple* slots, a simple slot being one that is neither transitive nor has any transitive subslots [Horrocks *et al.*, 1999] (note that the transitivity of a slot can be inferred, e.g., from the fact that the inverse of the slot is a transitive slot). An OIL `has-filler` constraint is equivalent to a set of has-value constraints where each individual i is transformed into a class expression of the form `one-of` i and each data value d is transformed into a datatype of the form `equal` d.

The complete mapping from OIL slot constraints to $\mathcal{SHIQ}(\mathcal{D})$ concepts is given in Figure 14.3, where SN is a slot or role name, C is a class expression or datatype, i is an individual and d is a data value (i.e., a string or an integer).

Class expressions One of the key features of OIL is that, in contrast to standard frame languages, class expressions are used instead of class names, e.g., in the list of super-classes, or in slot constraints. A class-expression is either a class name CN, an *enumerated-class*, a `slot-constraint`, a conjunction of class expressions

OIL	$\mathcal{SHIQ}(\mathcal{D})$
`slot-constraint` SN	$\top$
`has-value` $C_1 \ldots C_n$	$\sqcap \exists SN.\sigma(C_1) \sqcap \ldots \sqcap \exists SN.\sigma(C_n)$
`value-type` $C_1 \ldots C_n$	$\sqcap \forall SN.\sigma(C_1) \sqcap \ldots \sqcap \forall SN.\sigma(C_n)$
`max-cardinality` n C	$\sqcap \leqslant n\, SN.\sigma(C)$
`min-cardinality` n C	$\sqcap \geqslant n\, SN.\sigma(C)$
`cardinality` n C	$\sqcap \geqslant n\, SN.\sigma(C) \sqcap \leqslant n\, SN.\sigma(C)$
`has-filler` $i_1 \ldots d_n$	$\sqcap \exists SN.\sigma(\texttt{one-of } i_1) \sqcap \ldots \sqcap \exists SN.\sigma(\texttt{equal } d_n)$

Fig. 14.3. OIL to $\mathcal{SHIQ}(\mathcal{D})$ mapping (slot constraints).

OIL	$\mathcal{SHIQ}(\mathcal{D})$
`top`	$\top$
`thing`	$\top$
`bottom`	$\bot$
`and` $C_1 \ldots C_n$	$\sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
`or` $C_1 \ldots C_n$	$\sigma(C_1) \sqcup \ldots \sqcup \sigma(C_n)$
`not` C	$\neg\sigma(C)$
`one-of` $i_1 \ldots i_n$	$P_{i_1} \sqcup \ldots \sqcup P_{i_n}$

Fig. 14.4. OIL to $\mathcal{SHIQ}(\mathcal{D})$ mapping (class expressions).

(written `and` $C_1 \ldots C_n$), a disjunction of class expressions (written `or` $C_1 \ldots C_n$) or a negated class expression (written `not` C).

The class names `top`, `thing` and `bottom` have pre-defined interpretations: `top` and `thing` are interpreted as the most general class ($\top$), while `bottom` is interpreted as the inconsistent class ($\bot$). Note that `top` and `bottom` can just be considered as abbreviations for the class expressions (`or` C (`not` C)) and (`and` (C `not` C)) respectively (for some arbitrary class C).

An enumerated-class consists of a list of individual names, written `one-of` $C_1 \ldots C_n$. As already noted, this is not treated like the DL `one-of` constructor described in Chapter 2, but is mapped to a disjunction of disjoint primitive concepts corresponding to the individual names.

The complete mapping from OIL class expressions to $\mathcal{SHIQ}(\mathcal{D})$ concepts is given in Figure 14.4, where C is a class expression, i is an individual and P_i is the primitive concept corresponding to the individual i.

Datatypes In OIL slot constraints, datatypes and values can be used as well as or instead of class expressions and individuals. Datatypes can be either `integer` (i.e., the entire range of integer values), `string` (i.e., the entire range of string values), a subrange defined by a unary predicate such as `less-than` 10 or a Boolean combination of datatypes [Horrocks and Sattler, 2001].

The complete mapping from OIL datatypes to $\mathcal{SHIQ}(\mathcal{D})$ concepts is given in Figure 14.5, where d is a data value (an integer or a string), C is a datatype and

OIL	$\mathcal{SHIQ}(\mathcal{D})$
`min` d	$\geqslant_d$
`max` d	$\leqslant_d$
`greater-than` d	$>_d$
`less-than` d	$<_d$
`equal` d	$\geqslant_d \sqcap \leqslant_d$
`range` d_1 d_2	$\geqslant_{d_1} \sqcap \leqslant_{d_2}$
`and` $C_1 \ldots C_n$	$\sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
`or` $C_1 \ldots C_n$	$\sigma(C_1) \sqcup \ldots \sqcup \sigma(C_n)$
`not` C	$\neg\sigma(C)$

Fig. 14.5. OIL to $\mathcal{SHIQ}(\mathcal{D})$ mapping (datatypes).

OIL	$\mathcal{SHIQ}(\mathcal{D})$
`disjoint` $C_1 \ldots C_n$	$\sigma(C_1) \sqsubseteq \neg(\sigma(C_2) \sqcup \ldots \sqcup \sigma(C_n))$
	$\vdots$
	$\sigma(C_{n-1}) \sqsubseteq \neg\sigma(C_n)$
`covered` C `by` $C_1 \ldots C_n$	$\sigma(C) \sqsubseteq \sigma(C_1) \sqcup \ldots \sqcup \sigma(C_n)$
`disjoint-covered` C `by` $C_1 \ldots C_n$	$\sigma(C) \sqsubseteq \sigma(C_1) \sqcup \ldots \sqcup \sigma(C_n)$
	$\sigma(C_1) \sqsubseteq \neg(\sigma(C_2) \sqcup \ldots \sqcup \sigma(C_n))$
	$\vdots$
	$\sigma(C_{n-1}) \sqsubseteq \neg\sigma(C_n)$
`equivalent` $C_1 \ldots C_n$	$\sigma(C_1) \equiv \sigma(C_2), \ldots, \sigma(C_{n-1}) \equiv \sigma(C_n)$

Fig. 14.6. OIL to $\mathcal{SHIQ}(\mathcal{D})$ mapping (axioms).

$\geqslant_d$ (respectively $\leqslant_d$, $>_d$, $<_d$) is a unary predicate that returns true for all integers greater than or equal to (respectively less than or equal to, greater than, less than) d.

Axioms In addition to class definitions, OIL includes four kinds of axiom:

`disjoint` $C_1 \ldots C_n$ asserts that the class expressions $C_1 \ldots C_n$ are pairwise disjoint.

`covered` C `by` $C_1 \ldots C_n$ asserts that the class expression C is covered (subsumed) by the union of class expressions $C_1 \ldots C_n$.

`disjoint-covered` C `by` $C_1 \ldots C_n$ asserts that the class expression C is covered (subsumed) by the union of class expressions $C_1 \ldots C_n$, *and* that $C_1 \ldots C_n$ are pairwise disjoint.

`equivalent` $C_1 \ldots C_n$ asserts that the class expressions $C_1 \ldots C_n$ are equivalent.

The complete mapping from OIL axioms to $\mathcal{SHIQ}(\mathcal{D})$ axioms is given in Figure 14.6, where C is a class expression.

Slot definitions An OIL slot definition (`slot-def`) consists of a slot name SN followed by an optional documentation string and a slot description. A slot descrip-

OIL	$\mathcal{SHIQ}(\mathcal{D})$
`slot-def` SN	
`subslot-of` $RN_1 \ldots RN_n$	$SN \sqsubseteq RN_1, \ldots, SN \sqsubseteq RN_n$
`domain` $C_1 \ldots C_n$	$\exists SN.\top \sqsubseteq \sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
`range` $C_1 \ldots C_n$	$\top \sqsubseteq \forall SN.\sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
`inverse` RN	$SN^- \sqsubseteq RN$, $RN^- \sqsubseteq SN$
`properties transitive`	$SN \in \mathbf{R}_+$
`properties symmetric`	$SN \sqsubseteq SN^-$, $SN^- \sqsubseteq SN$
`properties functional`	$\top \sqsubseteq \leqslant 1\, SN$

Fig. 14.7. OIL to $\mathcal{SHIQ}(\mathcal{D})$ mapping (slot definitions).

$$
\begin{array}{rcl}
\mathsf{hasChild}^- & \sqsubseteq & \mathsf{isChildOf} \\
\mathsf{isChildOf}^- & \sqsubseteq & \mathsf{hasChild} \\
\mathsf{Woman} & \equiv & \mathsf{Person} \sqcap \mathsf{Female} \\
\mathsf{Man} & \equiv & \mathsf{Person} \sqcap \neg\mathsf{Woman} \\
\mathsf{Mother} & \equiv & \mathsf{Woman} \sqcap \exists\mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Father} & \equiv & \mathsf{Man} \sqcap \exists\mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Parent} & \equiv & \mathsf{Father} \sqcup \mathsf{Mother} \\
\mathsf{Grandmother} & \equiv & \mathsf{Mother} \sqcap \exists\mathsf{hasChild}.\mathsf{Parent} \\
\mathsf{MotherWithManyChildren} & \equiv & \mathsf{Mother} \sqcap \geqslant 3\, \mathsf{hasChild} \\
\mathsf{MotherWithoutDaughter} & \equiv & \mathsf{Mother} \sqcap \forall\mathsf{hasChild}.\neg\mathsf{Woman}
\end{array}
$$

Fig. 14.8. $\mathcal{SHIQ}(\mathcal{D})$ equivalent of the "family" ontology.

tion consists of an optional `subslot-of` component, with a list of one or more slot names, followed by a list of zero or more global slot constraints (e.g., `domain` and `range` constraints) and properties (e.g., `transitive` and `functional`).

The complete mapping from OIL class definitions to $\mathcal{SHIQ}(\mathcal{D})$ axioms is given in Figure 14.7, where SN and RN are slot or role names, C is a class expression and $\mathbf{R}_+$ is the set of $\mathcal{SHIQ}(\mathcal{D})$ transitive role names.

The mapping from OIL to $\mathcal{SHIQ}(\mathcal{D})$ has now been fully specified and we can illustrate, in Figure 14.8, the $\mathcal{SHIQ}(\mathcal{D})$ ontology corresponding to the OIL ontology from Figure 14.1.

14.3.1.2 XML and RDFS serialisations for OIL

The above language description uses OIL's "human readable" serialisation. This aids readability, but is not suitable for publishing ontologies on the web. For this purpose OIL is also provided with both XML and RDFS serialisations.

OIL's XML serialisation directly corresponds with the human readable form:

```
<ontology>
   <ontology-definitions>

      <slot-def>
         <slot name="hasChild"/>
         <inverse>
            <slot name="isChildOf"/>
         </inverse>
      </slot-def>

      <class-def type="defined">
         <class name="Woman"/>
         <subclass-of>
            <class name="Person"/>
            <class name="Female"/>
         </subclass-of>
      </class-def>

      <class-def type="defined">
         <class name="Man"/>
         <subclass-of>
            <class name="Person"/>
            <NOT>
               <class name="Woman"/>
            </NOT>
         </subclass-of>
      </class-def>

      <class-def type="defined">
         <class name="Mother"/>
         <subclass-of>
            <class name="Woman"/>
         </subclass-of>
         <slot-constraint>
            <slot name="hasChild"/>
            <has-value>
               <class name="Person"/>
            </has-value>
         </slot-constraint>
      </class-def>

   </ontology-definitions>
</ontology>
```

Fig. 14.9. OIL XML serialisation.

Figure 14.9 illustrates the XML serialisation of a fragment of the "family" ontology. A full specification and XML DTD can found in [Horrocks *et al.*, 2000a].

The RDFS serialisation is more interesting as it uses the features of RDFS both to capture as much as possible of OIL ontologies and to define a "meta-ontology" describing the structure of the OIL language itself. Figure 14.10 shows part of the RDFS description of OIL. The second and third lines contain XML namespace definitions that make the external RDF and RDFS definitions available for local use by preceding them with `rdf:` and `rdfs:` respectively. There then follows a "meta-ontology" describing (part of) the structure of OIL slot constraints.

The "meta-ontology" defines `hasPropertyRestriction` as an instance of RDFS `ConstraintProperty`[1] that connects an RDFS class (the property's `domain`) to an OIL property restriction (the property's `range`). A `PropertyRestriction` (slot constraint) is then defined as a kind of `ClassExpression`, with `HasValue` (an existential quantification) being a kind of `PropertyRestriction`. Properties `onProperty` and `toClass` are then defined as "meta-slots" of `PropertyRestriction` whose fillers will be the name of the property (slot) to be restricted and the restriction class expression. The complete description of OIL in RDFS, as well as a more detailed description of RDF and RDFS, can be found in [Horrocks *et al.*, 2000a].

Figure 14.11 illustrates the RDFS serialisation of a fragment of the "family" ontology. Note that most of the ontology consists of standard RDFS. For example, in the definition of `Woman` RDFS is used to specify that it is a `subClassOf` both `Person` and `Female`. Additional OIL specific vocabulary is only used where necessary, e.g., to specify that `Woman` is a defined class. The advantage of this is that much of the ontology's meaning would still be accessible to software that was "RDFS aware" but not "OIL aware".

14.3.2 DAML+OIL

DAML+OIL is similar to OIL in many respects, but is more tightly integrated with RDFS, which provides the only specification of the language and its only serialisation. While the dependence on RDFS has some advantages in terms of the re-use of existing RDFS infrastructure and the portability of DAML+OIL ontologies, using RDFS to completely define the structure of DAML+OIL is quite difficult as, unlike XML, RDFS is not designed for the precise specification of syntactic structure. For example, there is no way in RDFS to state that a restriction (slot constraint) should consist of exactly one property (slot) and one class.

The solution to this problem adopted by DAML+OIL is to define the semantics of the language in such a way that they give a meaning to any (parts of) ontologies that conform to the RDFS specification, including "strange" constructs such as slot constraints with multiple slots and classes. This is made easier by the fact that, unlike OIL, the semantics of DAML+OIL are directly defined in both a model theoretic and an axiomatic form (using KIF [Genesereth and Fikes, 1992]). The meaning given to strange constructs may, however, include strange "side effects".

[1] Property is the RDF name for a binary relation like a slot or role.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Property rdf:ID="hasPropertyRestriction">
    <rdf:type rdf:resource=
     "http://www.w3.org/2000/01/rdf-schema#ConstraintProperty"/>
    <rdfs:domain rdf:resource=
     "http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:range rdf:resource="#PropertyRestriction"/>
</rdf:Property>

<rdfs:Class rdf:ID="PropertyRestriction">
    <rdfs:subClassOf rdf:resource="#ClassExpression"/>
</rdfs:Class>

<rdfs:Class rdf:ID="HasValue">
    <rdfs:subClassOf rdf:resource="#PropertyRestriction"/>
</rdfs:Class>

<rdf:Property rdf:ID="onProperty">
    <rdfs:domain rdf:resource="#PropertyRestriction"/>
    <rdfs:range rdf:resource=
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Property>

<rdf:Property rdf:ID="toClass">
    <rdfs:domain rdf:resource="#PropertyRestriction"/>
    <rdfs:range rdf:resource="#ClassExpression"/>
</rdf:Property>

</rdf:RDF>
```

Fig. 14.10. Definition of OIL in RDFS.

For example, in the case of a slot constraint with multiple slots and classes, the semantics interpret this in the same way as a conjunction of all the constraints that would result from taking the cross product of the specified slots and classes, but with the added (and possibly unexpected) effect that all these slot constraints must have the same interpretation (i.e., are equivalent). Although OIL's RDFS based syntax would seem to be susceptible to the same difficulties, in the case of OIL there does not seem to be an assumption that any ontology conforming to the RDFS meta-description would be a valid OIL ontology—presumably ontologies containing unexpected usages of the meta-properties would be rejected by OIL processors as the semantics do not specify how these could be translated into $\mathcal{SHIQ}(\mathcal{D})$.

DAML+OIL's dependence on RDFS also has consequences for the decidability of the language. In OIL, the language specification states that the slots used in cardinality constraints can only be applied to simple slots (slots that are neither

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:oil="http://www.ontoknolwedge.org/oil/rdfschema">

<rdf:Property rdf:ID="hasChild">
   <oil:inverseRelationOf rdf:resource="#isChildOf"/>
</rdf:Property>
<rdf:Property rdf:ID="isChildOf"/>

<rdfs:Class rdf:ID="Woman">
   <rdf:type rdf:resource=
    "http://www.ontoknowledge.org/oil/rdfs-schema/#DefinedClass"/>
   <rdfs:subClassOf rdf:resource="#Person"/>
   <rdfs:subClassOf rdf:resource="#Female"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Man">
   <rdf:type rdf:resource=
    "http://www.ontoknowledge.org/oil/rdfs-schema/#DefinedClass"/>
   <rdfs:subClassOf rdf:resource="#Person"/>
   <rdfs:subClassOf>
      <oil:Not>
         <oil:hasOperand rdf:resource="#Woman"/>
      </oil:Not>
   </rdfs:subClassOf>
</rdfs:Class>

<rdfs:Class rdf:ID="Mother">
   <rdf:type rdf:resource=
    "http://www.ontoknowledge.org/oil/rdfs-schema/#DefinedClass"/>
   <rdfs:subClassOf rdf:resource="#Woman"/>
   <oil:hasPropertyRestriction>
      <oil:HasValue>
         <oil:onProperty rdf:resource="#hasChild"/>
         <oil:toClass rdf:resource="#Person"/>
      </oil:HasValue>
   </oil:hasPropertyRestriction>
</rdfs:Class>

</rdf:RDF>
```

Fig. 14.11. OIL RDFS serialisation.

transitive nor have transitive subslots). There is no way to capture this constraint in RDFS (although the language specification does include a warning about the problem), so DAML+OIL is theoretically undecidable. In practice, however, this may not be a very serious problem as it would be easy for a DAML+OIL processor to detect the occurrence of such a constraint and warn the user of the consequences.

Another effect of DAML+OIL's tight integration with RDFS is that the frame

```
<daml:ObjectProperty rdf:ID="hasChild">
  <daml:inverseOf rdf:resource="#isChildOf"/>
</daml:ObjectProperty>
<daml:Class rdf:ID="Woman">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Class rdf:about="#Female"/>
  </daml:intersectionOf>
</daml:Class>
<daml:Class rdf:ID="Man">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Class>
      <daml:complementOf rdf:resource="#Woman"/>
    </daml:Class>
  </daml:intersectionOf>
</daml:Class>
<daml:Class rdf:ID="Mother">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Woman"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasChild"/>
      <daml:hasClass rdf:resource="#Person"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>
```

Fig. 14.12. DAML+OIL ontology serialisation.

structure of OIL's syntax is much less evident: a DAML+OIL ontology is more DL-like in that it consists largely of a relatively unstructured collection of subsumption and equality axioms. This can make it more difficult to use DAML+OIL with frame based tools such as PROTÉGÉ [Grosso *et al.*, 1999] or OILED [Bechhofer *et al.*, 2001b] because the axioms may be susceptible to many different frame-like groupings [Bechhofer *et al.*, 2001a].

From the point of view of language constructs, the differences between OIL and DAML+OIL are relatively trivial. Although there is some difference in "keyword" vocabulary, there is usually a one to one mapping of constructors, and in the cases where the constructors are not completely equivalent, simple translations are possible. For example, DAML+OIL restrictions (slot constraints) use `has-class` and `to-class` where OIL uses `ValueType` and `HasValue`, and while DAML+OIL has no direct equivalent to OIL's covering axioms, the same effects can be achieved using a combination of (disjoint) union and `subClass`. The similarities can clearly be seen in Figure 14.12, which illustrates the DAML+OIL version of the "family" ontology fragment from Figure 14.9.

The treatment of individuals in DAML+OIL is, however, very different from that in OIL. In the first place, DAML+OIL relies wholly on RDF for ABox assertions, i.e., axioms asserting the type (class) of an individual or a relationship between a pair of individuals. In the second place, DAML+OIL treats individuals occurring in the ontology (in `oneOf` constructs or `hasValue` restrictions) as true individuals (i.e., interpreted as single elements in the domain of discourse) and not as primitive concepts as is the case in OIL (see Chapter 2). Moreover, there is no unique name assumption: in DAML+OIL it is possible to explicitly assert that two individuals are the same or different, or to leave their relationship unspecified.

This treatment of individuals is very powerful, and justifies intuitive inferences that would not be valid for OIL, e.g., that persons all of whose countries of residence are Italy are kinds of person that have at most one country of residence:

$$\mathsf{Person} \sqcap \forall \mathsf{residence}.\{\mathsf{Italy}\} \sqsubseteq \leqslant 1\, \mathsf{residence}$$

Unfortunately, the combination of individuals with inverse roles is so powerful that no "practical" decision procedure (for satisfiability/subsumption) is currently known, and there is no implemented system that can provide sound and complete reasoning for the whole DAML+OIL language. In the absence of inverse roles, however, a tableaux algorithm has been devised [Horrocks and Sattler, 2001], and in the absence of individuals DAML+OIL ontologies can exploit implemented DL systems via a translation into $\mathcal{SHIQ}$ similar to the one described for OIL. It would, of course, also be possible to translate DAML+OIL ontologies into $\mathcal{SHIQ}$ using the disjoint primitive concept interpretation of individuals adopted by OIL, but in this case reasoning with individuals would not be sound and complete with respect to the semantics of the language.

14.3.2.1 DAML+OIL datatypes

The initial release of DAML+OIL did not include any specification of datatypes. However, in the March 2001 release,[1] the language was extended with arbitrary datatypes from the XML Schema type system,[2] which can be used in restrictions (slot constraints) and range constraints. As in $\mathcal{SHOQ}(\mathcal{D})$ [Horrocks and Sattler, 2001], a clean separation is maintained between instances of "object" classes (defined using the ontology language) and instances of datatypes (defined using the XML Schema type system). In particular, it is assumed that the domain of interpretation of object classes is disjoint from the domain of interpretation of datatypes, so that an instance of an object class (e.g., the individual Italy) can never have the same interpretation as a value of a datatype (e.g., the integer 5), and that the set of object properties (which map individuals to individuals) is disjoint from the set of datatype properties (which map individuals to datatype values).

[1] `http://www.daml.org/2001/03/daml+oil-index.html`
[2] `http://www.w3.org/TR/xmlschema-2/#typesystem`

The disjointness of object and datatype domains was motivated by both philosophical and pragmatic considerations:

- Datatypes are considered to be already sufficiently structured by the built-in predicates, and it is, therefore, not appropriate to form new classes of datatype values using the ontology language [Hollunder and Baader, 1991b].
- The simplicity and compactness of the ontology language are not compromised—even enumerating all the XML Schema datatypes would add greatly to its complexity, while adding a theory for each datatype, even if it were possible, would lead to a language of monumental proportions.
- The semantic integrity of the language is not compromised—defining theories for all the XML Schema datatypes would be difficult or impossible without extending the language in directions whose semantics may be difficult to capture in the existing framework.
- The "implementability" of the language is not compromised—a hybrid reasoner can easily be implemented by combining a reasoner for the "object" language with one capable of deciding satisfiability questions with respect to conjunctions of (possibly negated) datatypes [Horrocks and Sattler, 2001].

From a theoretical point of view, this design means that the ontology language can specify constraints on data values, but as data values can never be instances of object classes they cannot apply additional constraints to elements of the object domain. This allows the type system to be extended without having any impact on the object class (ontology) language, and vice versa. Similarly, reasoning components can be independently developed and trivially combined to give a hybrid reasoner whose properties are determined by those of the two components; in particular, the combined reasoner will be sound and complete if both components are sound and complete.

From a practical point of view, DAML+OIL implementations can choose to support some or all of the XML Schema datatypes. For supported data types, they can either implement their own type checker/validater or rely on some external component (non-supported data types could either be trapped as an error or ignored). The job of a type checker/validater is simply to take zero or more data values and one or more datatypes, and determine if there exists any data value that is equal to every one of the specified data values and is an instance of every one of the specified data types.

14.4 Summary

It has long been realised that the web would benefit from more structure, and it is widely agreed that ontologies will play a key role in providing this structure. Description logics have made important contributions to research in this area, ranging from formal foundations and early web applications through to the development of description logic based languages designed to facilitate the development and deployment of web ontologies. OIL and its successor DAML+OIL are two such ontology languages, specifically designed for use on the web; they exploit existing web standards (XML, RDF and RDFS), adding the formal rigor of a description logic and the ontological primitives of object oriented and frame based systems.

This combination of features has proved very attractive, and DAML+OIL has already been widely adopted. At the time of writing, the DAML ontology library contains over 175 ontologies, and DAML crawlers have found millions of DAML+OIL markup statements in documents. Possibly more important, however, is that some major efforts have committed to encoding their ontologies in DAML+OIL. This has been particularly evident in the bio-ontology domain, where the Bio-Ontology Consortium has specified DAML+OIL as their ontology exchange language, and the Gene Ontology [The Gene Ontology Consortium, 2000] is being migrated to DAML+OIL in a project partially funded by GlaxoSmithKline Pharmaceuticals in cooperation with the Gene Ontology Consortium.

There has also been significant progress in the development of tools supporting DAML+OIL. Several DAML+OIL ontology editors are now available including Manchester University's OilEd (which incorporates reasoning support from the FaCT system) [Bechhofer *et al.*, 2001b], Protégé [Grosso *et al.*, 1999] and OntoEdit [Staab and Maedche, 2000]. At Stanford University, a combination of Ontolingua, Chimaera and JTP (Java Theorem Prover) are being used to provide editing, evolution, maintenance, and reasoning services for DAML+OIL ontologies [McGuinness *et al.*, 2000b; 2000a]. Commercial endeavors are also supporting DAML+OIL. Network Inference Limited, for example, have developed a DAML+OIL reasoning engine based on their own implementation of a DL reasoner.

What of the future? The development of the semantic web, and of web ontology languages, presents many opportunities and challenges for description logic research. A "practical" (satisfiability/subsumption) algorithm for the full DAML+OIL language has yet to be developed, and even for OIL, it is not yet clear that sound and complete DL reasoners can provide adequate performance for typical web applications. It is also unclear how a DL system would cope with the very large ABoxes that could result from the use of ontologies to add semantic markup to (large numbers of) web pages. DL researchers are also beginning to address new inference problems that may be important in providing reasoning services for the semantic web, e.g., querying [Rousset, 1999a; Calvanese *et al.*, 1999a; Horrocks and Tessaris, 2000], matching [Baader *et al.*, 1999a] and computing least common subsumers and most specific concepts [Cohen *et al.*, 1992; Baader and Küsters, 1998; Baader *et al.*, 1999b].

Finally, the developers of both OIL and DAML+OIL always understood that a single language would not be adequate for all semantic web applications—OIL even gave a name (Heavy OIL) to an as yet undefined extension of the language—and extensions up to (at least) full first order logic are already being discussed. Clearly, most of these extended languages will be undecidable. Description Logics research can, however, still make important contributions, e.g., by investigating the boundaries of decidability, identifying decidable subsets of extended languages and developing decision procedures. DL implementations can also play a key role, both as reasoning engines for the core language and as efficient components of hybrid reasoners dealing with a variety of language extensions.

Acknowledgements

We would like to thank Jérôme Euzenat and Frank van Harmelen for their insightful comments on a previous version of the paper. All remaining errors are, of course, our own.