Ontology Representation : design patterns and ontologies that make sense

Hoekstra, R.J.

Chapter 3

Semantic Web

> "He could spell his own name WOL, and he could spell Tuesday so that you knew it wasn't Wednesday but his spelling goes all to pieces over delicate words like measles and buttered toast."
>
> *About Owl, in 'Winnie the Pooh' by A.A.Milne*

3.1 Introduction

In November 2001, the WebOnt working group set out to develop a new Web Ontology Language (OWL, Bechhofer et al. (2004)) to be used for knowledge representation in a form that it could be shared across the web.[1] The working group was tasked with the development of:[2]

> "A Web ontology language, that builds on current Web languages that allow the specification of classes and subclasses, properties and subproperties (such as RDFS), but which extends these constructs to allow more complex relationships between entities including: means to limit the properties of classes with respect to number and type, means to infer that items with various properties are members of a particular class, a well-defined model of property inheritance, and similar semantic extensions to the base languages."
>
> WebOnt WG Charter

The Web Ontology Language is tightly interwoven with the development of the *semantic web*, first described in Berners-Lee (1999); Berners-Lee et al. (2001). The purpose of the Semantic Web is to advance the machine interpretability

[1] The working group disliked the proper acronym "WOL" and decided to call the language "OWL". This decision was backed by the extraordinary abilities of the wise friend of Winnie the Pooh: Owl.

[2] `http://www.w3.org/2002/11/swv2/charters/WebOntologyCharter`

of information stored on the web. Not just for improving the accessibility and communication of this information to humans, but primarily for *knowledge* exchange between *automated web services*. More importantly, semantic web technology should enable open and unattended sharing of this knowledge. A web-based knowledge representation language should take into account the distributed nature of the web, and is necessarily subject to other constraints than a language that is not exposed in that way: how does this trade off work out in practice?

In this chapter I give an overview of the technical characteristics of semantic web languages, and the trade-offs they imply for web-based knowledge representation. Central to this discussion is the Web Ontology Language, which is in many ways a natural step up the evolutionary ladder for the terminological knowledge representation languages of the 80's and 90's. Although it can be said that a significant body of documentation on OWL already exists, this has either the nature of reference guide or overview (Bechhofer et al., 2004; McGuinness and van Harmelen, 2004), a technical specification (Patel-Schneider et al., 2004) or rather entry-level guides and walkthroughs (Antoniou and van Harmelen, 2004; Smith et al., 2004; Horridge et al., 2007). This chapter aims to lay bare the rationale and limitations of the language features of OWL, both practical (Antoniou and van Harmelen, 2003) and theoretical (Horrocks and Patel-Schneider, 2003). This view plays a central role in the description of *design patterns* in Chapter 7.

The fact that OWL is called a web *ontology* language, obfuscates rather than emphasises its main character as a highly expressive description logic. For all practical purposes, this chapter will simply use the term 'ontology' to denote a terminological knowledge representation (or domain theory) expressed using the web ontology language. Chapter 4 discusses the ambiguity of the term 'ontology', and sheds light on the historical reasons for giving this language the name "OWL".

3.1.1 The Web

The web as it exists from roughly 1993 on is not much more than a network of inter linked pages. For sure, its success is very much based on this lightweight approach: it uses a simple, networked architecture which is easily extensible and immune to incorrect or incomplete data, links and mark-up. Information is presented in a human understandable way: by means of texts, pictures, layout etc. Legacy web languages such as HTML[3] and CSS[4] are specifically targeted to facilitate the disclosure of information in a way suitable for human consumption. In a way, more recent developments such as Ajax[5] take this approach even further by making web pages more responsive to human interaction.

The primary means of navigating the web is by following hyperlinks between pages, form filling and the use of search engines such as Google and Yahoo Search. These search engines use a combination of natural language processing (NLP), network analysis and hit-counts to determine the rank of a page in search results. Nonetheless, still even the best engines suffer from

[3]Hypertext Markup Language, `http://www.w3.org/html/`
[4]Cascading Style Sheets, `http://www.w3.org/Style/CSS/`
[5]Asynchronous Javascript and XML

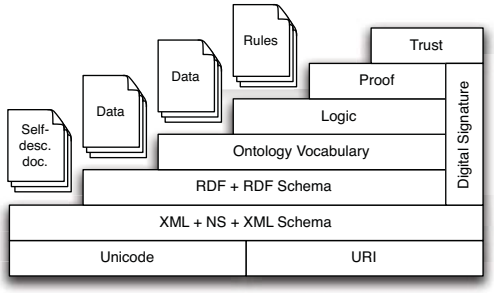

Figure 3.1: The SemanticWeb layer cake

either a high recall (low precision) or low or no recall at all (Antoniou and van Harmelen, 2004). Both search queries and results are language dependent, i.e. a search using one language will not reveal relevant results in another. Furthermore search results are single webpages, whereas information relevant to the query might be distributed across many. It is therefore unsurprising that companies spend huge sums to improve the visibility of their pages in these search engines. According to a Forrester report, spending will increase from today's 4.5 billion to 8.1 billion euros in 2012.[6]

A presentation-oriented way of publishing information on the web leaves room for improvement as it is only partially suited for machine interpretation. In other words, it is very hard for machines to find out what a page is *about* by looking at its structure. In many ways this issue is similar to the problems a visually impaired person experiences when browsing the web (Bechhofer et al., 2006). Considering that most webpages are generated from a structured data source it is at least odd that search engines and other crawlers have to extract meaning in this very roundabout way. Why not present information on the web in a way which *is* directly accessible to machines?

3.2 Groundwork

The Semantic Web (Berners-Lee et al., 2001) is a non-intrusive extension of the current web and adds different 'layers' of machine processable semantics. The extension is non-intrusive because each layer builds on another layer without modifying it, see Figure 3.1. Semantics on the web is organised using the same philosophy that underlies current web technologies: transparent heterogeneous networks of linked resources, similar to the Semantic Nets of Section 2.2.3. The first two layers of this cake introduce standards for *character encoding* and *identification*, and a modular mechanism for describing the *structure* of data.

[6] "Europe's Search Engine Marketing Investment Exceeds EUR 8 Billion In 2012", by Mary Beth Kemp, `http://www.forrester.com/Research/Document/Excerpt/0,7211,42707,00.html`

Character Encoding

Characters should be encoded in a uniform way in order for strings to be interpreted the same way across the globe. This holds for presentation-oriented mechanisms – e.g. browsers should be able to display both chinese and western characters – and extensive data manipulation mechanisms alike. Unicode[7] is a standard adopted by industry that enables computers to consistently represent and manipulate text expressed using a wide range of writing systems. Amongst others, it defines several mappings from (subsets of) some 100k characters onto various encodings using the *Unicode Transformation Format* (UTF). The most well-known of these is UTF-8, an 8 bit encoding with variable-width that is backwards compatible with the older ASCII format.[8]

Identification

By adopting *Universal Resource Identifiers* (URIs)[9] as means for unique identification of resources on the web, anyone can make statements about other statements similar to the way in which one can now hyperlink from one HTML page to another using a Universal Resource *Location* (URL). In fact, every URL is also a URI (but not vice versa):[10] a URL is a URI used as a location. Alternatively a URI can be used as a *Universal Resource Name* (URN).[11] Note that URI's, including URN's, attach a meaningful machine interpretable *identifier* to resources on the web. They are explicitly not intended to be used as human readable *names*. URI's are structured as follows:

URI = scheme ":" hier-part ["?" query] ["#" profile]

Namespaces

A resource on the web is often specified 'within' a *namespace*. A namespace is essentially a collection of URI's in which every resource name is locally unique. The URI of a namespaced resource is composed of a namespace name, and a local name. Usually, the profile part of a URI is used to contain the local name and the composition of scheme and hier-part is used to denote the namespace name. This mechanism can be used to provide *authority* to a namespace, as namespace names can correspond to domain names on the internet. This is a loosely coupled form of 'trust', as the only way to enforce this authority is by dereference-ability of URI's to a location. Namespaces can be abbreviated using so-called *namespace prefixes*, which typically abbreviate the scheme and hier-part to an intelligible shorthand.

[7] See `http://www.unicode.org/`

[8] ASCII: American Standard Code for Information Interchange

[9] The URI specification is defined in RFC 3986, see `http://gbiv.com/protocols/uri/rfc/rfc3986.html`

[10] This simple relation has been the source of much confusion, as many treat URI's as URL's, i.e. by dereferencing from a URI to a URL.

[11] It is often thought that URN's should always use the 'urn' scheme, where syntactic parts of the name are delimited by colons. In fact, *any* URI can be interpreted as either a name or a location, it is the interpretation that turns a URI into a URN or a URL respectively. However, URN's specified using a 'urn' scheme cannot be dereferenced to a URL without a defined mapping.

Structured Data

The *Extensible Markup Language* (XML)[12] is a flexible way to explicitly encode data in a nested datastructure. It is a subset of SGML,[13] an industry standard meta language for defining exchangeable machine processable document formats. XML diverges from SGML in general by its orientation towards the exchange of *data* on the *web*, and was designed to be compatible with HTML, e.g. XHTML 1.0 Strict is an XML version of HTML and supersedes older versions. Extensibility of XML is provided by the XML Schema[14] language that can be used to describe restrictions on the structure of XML encoded data within a particular namespace; XML Schema is itself an XML dialect. It provides a fixed set of datatypes, such as integers, strings, date, that can be used by XML parsers when interpreting a document.

Important to note, however, is the fact that although the *structure* of some data can be expressed in XML according to some schema, the way in which this data should be interpreted, i.e. its *semantics*, is externally defined (if at all). Even when we *do* commit to a particular interpretation of the nested structure of XML, the least we can say is that it does not convey very expressive semantics.

3.3 Lightweight Semantics

"A little semantics goes a long way"

James Hendler

The first step towards the Semantic Web is the specification of a way to add simple, relatively lightweight metadata to documents on the web. Currently, the de facto metadata representation languages on the semantic web are the Resource Description Framework (RDF) and the RDF Schema vocabulary description language extension to RDF. Contrary to other knowledge structuring languages, such as Topic Maps (Biezunski et al., 1999)[15] or even UML, which have XML syntaxes and can thus be used to publish metadata on the web, RDF and RDF Schema are *web languages*. That is, the structure of the RDF language is designed to mimic the way in which information is stored on the web. This way, metadata can be distributed across multiple documents or locations; and is extensible as any RDF resource can point directly to other RDF resources in the same way that HTML documents can.

RDF is often criticised as being too technical, having unwieldy semantics and verbose syntax, and at the same time providing only a limited vocabulary of primitives for (lexical) knowledge organisation. These attributes would

[12] See `http://www.w3.org/TR/xml/`

[13] SGML: Standard Generalized Markup Language

[14] See `http://www.w3.org/TR/xmlschema-1/` (structure) and `http://www.w3.org/TR/xmlschema-2/` (datatypes)

[15] See `http://www.topicmaps.org/`, the XML syntax for Topic Maps is XTM, `http://www.topicmaps.org/xtm/index.html`

make it a less likely candidate for intuitive knowledge representation. In part, this can be overcome because RDF is in general more expressive than Topic Maps – RDF descriptions have a higher granularity – and the semantics of both languages can be mapped to each other (Pepper et al., 2006).[16] A related initiative is the Simple Knowledge Organisation System (SKOS) described in Miles and Bechhofer (2008) that defines an RDF data model for expressing the taxonomies, classification structures and thesauri used in many applications to structure and organise knowledge and information such as WordNet.[17]

In fact, there is a subtle distinction between the purpose of languages such as Topic Maps and SKOS, and that of RDF. The former are designed to describe what documents and other information sources (such as database entries) are *about*. Ultimately, these languages are designed to describe indexes on information sources akin to the old fashioned card indexes used in libraries. Both SKOS and Topic Maps are typically used to capture lexical categories, *terms*, rather than semantic categories, *concepts*. In fact, the SKOS specification is very explicit about its purpose and states: "SKOS is not a knowledge representation language" (Miles and Bechhofer, 2008). Although RDF is similarly designed to state facts about information sources, these statements are not necessarily intended to capture (only) the information content of a source: it can be used to express *more*.

3.3.1 RDF: The Resource Description Framework

RDF is a lightweight and flexible way to represent metadata on the web. It is a simple assertional language that defines a way to make statements *about* things on the web. These statements take the form of subject, predicate, object triples $< s, p, o >$ a syntactic variant of traditional binary predicates, e.g. $p(s, o)$. The assertion of such a triple is defined to mean that predicate p is a relation between s and o. Each part of the triple, i.e. each RDF name, denotes a *resource*.

How a name is treated depends on its syntactic form: URI references are treated as logical constants, but plain *literals* of the form "literal value" denote themselves and have a fixed meaning. A literal that is typed by an XML Schema datatype, e.g. "1"^^ xsd:Integer denotes the value resulting from a mapping of the literal value (the enclosed character string) by the datatype. For instance, the literal "1"^^ xsd:Integer denotes the integer value 1, whereas "1" simply denotes the exact string of characters "1", including the quotes. A resource that has a name which is a URI reference, denotes the entity that can be identified by means of the URI. It *does not* denote the URI itself; nor does it necessarily denote the entity found at the location when the URI is dereferenced as if it were a URL. In other words, a RDF resource can be *anything*, and does not have to exist on the web. Furthermore, a URI cannot be used to identify multiple entities.

A collection of interconnected RDF triples constitutes a directed *graph*, where nodes are the subjects and objects of assertions, and properties are edges. Figure 3.2 shows an example graph that expresses the phrase "Joost Breuker supervises Rinke Hoekstra". In this example, the names "Joost Breuker" and

[16] See e.g. the working group note of the RDF/Topic Maps Interoperability Task Force (RDFTM) at http://www.w3.org/TR/rdftm-survey/ for an overview.

[17] See http://www.w3.org/2004/02/skos/. The SKOS data model itself is expressed in OWL Full.

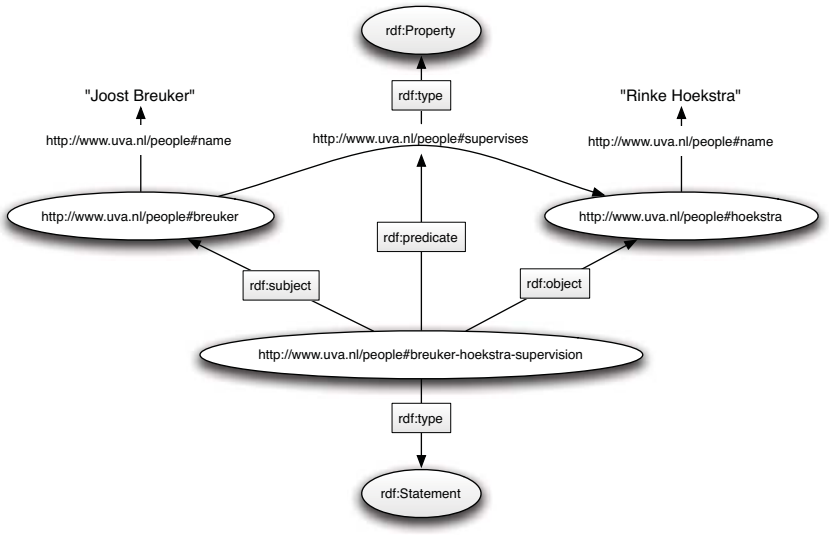

Figure 3.2: An RDF graph representation of the phrase "Joost Breuker supervises Rinke Hoekstra"

"Rinke Hoekstra" are literal values, connected to resources that *are* the respective persons. These resources are identified by URI's in the `http://www.uva.nl/people` namespace: the URI `http://www.uva.nl/people#breuker` denotes Joost Breuker, the actual person. In the following I will abbreviate URIs in the `http://www.uva.nl/people` namespace using the prefix 'uva'.

The supervision relation is denoted by the predicate uva:supervises that holds between the two persons. Not only can this predicate itself be addressed as a resource, as in uva:supervises rdf:type rdf:Property, but any triple as a whole can be *reified*, and explicitly named. In RDF, reification is expressed using the rdf:Statement construct. A resource of type rdf:Statement can explicitly refer to the subject, predicate and object of some property relation using the rdf:subject, rdf:predicate and rdf:object properties, respectively. For instance, the resource uva:breuker-hoekstra-supervision in Figure 3.2 reifies the uva:supervises relation by explicitly pointing to its relata. Note, however, that although the existence of a relation indicates the existence of its reification, an rdf:Statement by itself does not mean that the corresponding relation holds as well (see also Section 7.3.3). In the current example, the assertion of the uva:breuker-hoekstra-supervision statement and its relata does allow us to infer the triple uva:breuker uva:supervises uva:hoekstra.

RDF graphs can be stored in different formats. Most commonly this will be some plain text file, which means that the graph needs to be *serialised*, i.e. 'flattened' or deflated to fit the a sequential order of characters in a file. RDF serialisations are order independent because the order in which triples are added to a file depends on an arbitrary choice for which node is serialised first. There exist three official serialisation syntaxes for RDF: RDF/XML, N-Triple and more recently the Terse RDF Triple Language (Turtle).[18]

[18]See `http://www.w3.org/TR/rdf-syntax-grammar/`, `http://www.w3.org/`

The RDF/XML syntax has the advantage of being a native XML format, though parsing it can be hard as the native order-dependent tree model of XML documents gets in the way of the RDF graph model. Secondly, this makes the syntax notoriously verbose. For instance, the RDF graph of Figure 3.2 is serialised in RDF/XML as follows:[19]

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:uva="http://www.uva.nl/people#"
      xml:base="http://www.uva.nl/people">
   <rdf:Property rdf:ID="supervises"/>
   <rdf:Property rdf:ID="name"/>
   <rdf:Description rdf:about="#breuker" >
      <uva:supervises rdf:resource="#hoekstra"/>
      <uva:name>"Joost Breuker"</uva:name>
   </rdf:Description>
   <rdf:Description rdf:about="#hoekstra" >
      <uva:name>"Rinke Hoekstra"</uva:name>
   </rdf:Description>
   <rdf:Statement rdf:ID="breuker-hoekstra-supervision">
      <rdf:subject rdf:resource="#breuker"/>
      <rdf:predicate rdf:resource="#supervises"/>
      <rdf:object rdf:resource="#hoekstra"/>
   </rdf:Statement>
</rdf:RDF>
```

RDF/XML uses the rdf:about property to state that some rdf:Description concerns the resource indicated by the URI reference. The rdf:resource property connects the predicate of a relation to its object, e.g. the object of the rdf:subject relation in the statement uva:breuker- hoekstra- supervision is the resource uva: breuker.[20] Provided that the type of some resource is known, as is the case with the uva:supervises and uva:name properties, we can directly state the properties of that resource under an element of its type and a rdf:ID attribute that gives the resource's URI. For instance, the serialisation above states that the uva:supervises resource is of type rdf:Property. An equivalent serialisation that does not explicitly introduce the resource, but rather adds information *about* it, is:

`TR/2004/REC-rdf-testcases-20040210/#ntriples` and `http://www.w3.org/TeamSubmission/turtle/`, respectively.

[19] Note that the xml:base attribute is used to abbreviate some of the URIs.

[20] Namespace abbreviation can be a bit confusing because of the intermixed use of RDF URI references and xml:ID datatypes for the identifiers. The rdf:ID attribute is an xml:ID and its value is automatically interpreted by an XML parser as an identifier within the base (or default) namespace. This means that an RDF parser does not have to deal with whether an rdf:ID is a full URI or just the fragment, as it will be presented as a full URI of the form *<namespace>+#+<local name>*. The rdf:about and rdf:resource properties, on the other hand, do not specify the identity of the XML element they are located at, but rather *point towards* some other resource. If the URI reference is not fully specified, an RDF parser will have to resolve the URI, and it does this by simply concatenating the value of the xml:base attribute and the URI fragment identifier: *xml:base+<fragment identifier>*. Hence the necessity to prefix the name with a pound character (#).

```
<rdf:Description rdf:about="#supervises">
  <rdf:type
    rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Description>
```

Turtle was developed as a subset of the N3 language,[21] and has a much more readable syntax than RDF/XML. It is similar to N-Triple, in which all triples in the graph are spelled-out, but it allows several shorthand notations. For instance, the omission of the object of a triple in several consecutive statements that are separated by semicolons, and the reserved word a that replaces rdf:type. The Turtle serialisation of the RDF graph in Figure 3.2 is as follows:

```
@prefix uva: <http://www.uva.nl/people#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .


uva:breuker
    uva:name "\"Joost Breuker\"" ;
    uva:supervises uva:hoekstra .

uva:breuker-hoekstra-supervision
    a rdf:Statement ;
    rdf:object uva:hoekstra ;
    rdf:predicate uva:supervises ;
    rdf:subject uva:breuker .

uva:hoekstra
    uva:name "\"Rinke Hoekstra\"" .

uva:name
    a rdf:Property .

uva:supervises
    a rdf:Property .
```

3.3.2 The RDF Schema Vocabulary Description Language

Although RDF can already be used to describe quite complex graph structures, it provides little in the way of semantics and automatic inferencing. For instance, there is no standard way to say that both uva:breuker and uva:hoekstra are persons, or that the uva:supervises relation is a relation only between persons. RDF Schema (RDFS) extends RDF with basic primitives that do allow us to express such more generic knowledge.

In fact, most of these primitives are already commonplace in the semantic networks of the 1970s. RDFS introduces the notion of classes (rdfs:Class) and transitive subclass relations (rdfs:subClassOf) which allow to describe taxonomies. Individual resources can be said to belong to a class using the rdf:type property. Under RDFS semantics, the type of a resource is inherited over the rdfs:subClassOf relation: resources belonging to a class belong to all its super

[21] See `http://www.w3.org/DesignIssues/Notation3.html`. N3 allows for several non-RDF constructs such as a forAll loop, rules and paths.

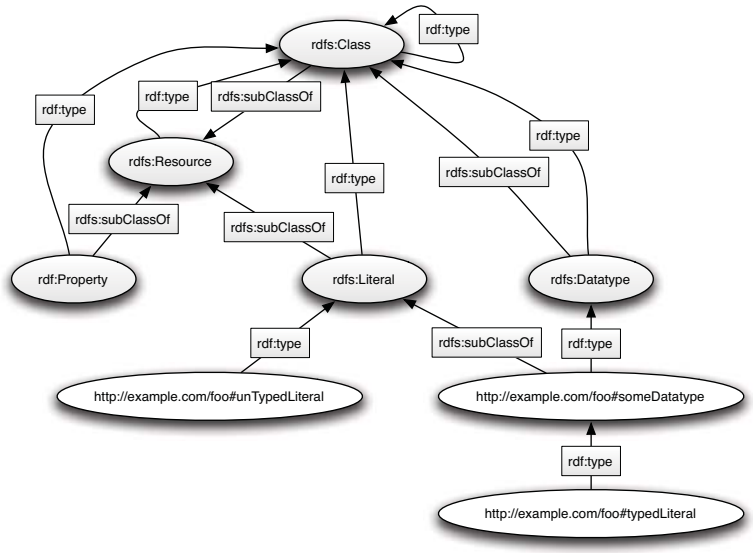

Figure 3.3: Overview of relations between RDFS classes.

classes. The resource rdfs:Class is defined as the set of all classes: all classes in RDFS are of the type rdfs:Class. The class rdfs:Resource is the set of all resources, and is therefore by default the super class of all resources of type rdfs:Class. All RDF literals are members of the class rdfs:Literal. Typed literals are consequently instances of the subclass of rdfs:Literal defined by their datatype. Each XML Schema datatype supported by RDF is a member of the class rdfs:Datatype. The rdf:Property, although still in the RDF namespace, is defined as the class of all properties. The relations between the various RDFS classes are shown in Figure 3.3.

The transitive rdfs:subPropertyOf property can be used to specify that two resources related by one property (the sub property) are also related by another property (the super property). In RDF Schema, any resource of the type rdf:Property can be assigned a domain and range. Domain and range specifications are inherited over the rdfs:subPropertyOf relation. If any two resources are related by a rdfs:domain or rdfs:range property, then the subject of that relation is an instance of rdf:Property and the object an instance of rdfs:Class. If that property then is used to relate two other resources, their type is inferred to be of the classes specified by the domain and range. Multiple domain and range restrictions are interpreted as intersections, i.e. class membership is inferred for all classes in the domain or range, respectively.[22]

To given an example, given just the following three triples:

[22]The rdf:type property is a special property and holds between instances and classes, instead of only classes. It therefore does not have a rdfs:domain property

```
uva:supervises
    rdfs:domain uva:Person ;
    rdfs:range uva:Person .

uva:breuker
    uva:supervises uva:hoekstra .
```

we can infer that uva:supervises is a property, uva:Person is a class and that uva:breuker and uva:hoekstra both belong to that class:

```
uva:supervises
    a rdf:Property .

uva:Person
    a rdfs:Class .

uva:breuker
    a uva:Person .

uva:hoekstra
    a uva:Person .
```

The properties rdfs:label and rdfs:comment can be used to annotate resources in human readable form, using multiple languages (by means of language tags). Values of an rdfs:label property are intended to give a human readable name alternative to the URI of a resource. The range of these properties is the xsd:string datatype:

```
uva:name
    a rdf:Property;
    rdfs:label "Name"@en;
    rdfs:label "Nom"@fr;
    rdfs:label "Name"@de;
    rdfs:label "Naam"@nl;
    rdfs:comment "This property is used to specify a person's name."@en;
rdfs:comment "Dit property wordt gebruikt voor de naam van een persoon."@nl.
```

Meta-Modelling in RDFS

RDFS has a non-fixed meta modelling architecture; it can have an infinite number of class layers because rdfs:Resource is both an instance and a super class of rdfs:Class, which makes rdfs:Resource a member of its own subset (Nejdl et al., 2000). All classes (including rdfs:Class itself) are instances of rdfs:Class, and every class is the set of its instances. There is no restriction on defining sub classes of rdfs:Class itself, nor on defining sub classes of instances of instances of rdfs:Class and so on. This is problematic as it leaves the door open to class definitions that lead to Russell's paradox (Pan and Horrocks, 2002). The Russell paradox follows from a comprehension principle built in early versions of set theory (Horrocks et al., 2003). This principle stated that a set can be constructed of the things that satisfy a formula with one free variable. In fact, it introduces the possibility of a set of all things that do not belong to itself: $\{x|x \notin x\}$ given the formula $x \notin x$. We cannot determine whether the class x is a member of

itself: this only holds if and only if it is not a member of itself.

Meta modelling of the type sanctioned by RDFS provides no means to exclude the definition of such classes from extensions of the RDFS semantics. For instance, a straightforward semantic extension of RDFS that allows qualified cardinality restrictions (as the DL's $\mathcal{SHIQ}(D)$ and $\mathcal{SHROIQ}(\mathcal{D})$ do, cf. Section 4.2 and Section 3.5.2, respectively) would allow us to define a class as an instance of itself, or even add a cardinality constraint of 0 on the rdf:type property (Pan and Horrocks, 2003).

In RDFS, the reserved properties rdfs:subClassOf, rdf:type, rdfs:domain and rdfs:range are used to define both the other RDFS modelling primitives themselves and the models expressed using these primitives. In other words, there is no distinction between the meta-level and the domain. In terms of Figure 2.5 in Section 2.3.1, this means that RDFS conflates the language and representation columns.

One example is the circular definition of rdfs:subClassOf and rdfs:Class; rdfs:Class is defined as the rdfs:subClassOf rdfs:Resource, but rdfs:subClassOf is defined as a relation between resources of the type rdfs:Class. Some of the promiscuity of RDFS primitives was resolved by specifying a model theoretic semantics (Hayes, 2004, RDFS MT).[23] However, RDFS MT included the inherent problems with a non-fixed meta modelling architecture (Pan and Horrocks, 2003; Motik, 2007).

In a nutshell, RDFS has its appeals as a relatively lightweight means to add additional semantics to RDF models, but it falls short in many ways as a language for specifying knowledge on the web. As knowledge representation language the RDFS is still quite weak. Other than domain and range restrictions it has no means to describe the requirements for class-membership needed for concise definitions, such as necessary & sufficient conditions, cardinality constraints and negation. For instance, it can be useful to express that two classes are *disjoint*, e.g. to distinguish between animals and plants: no animal can be a plant and vice versa;[24] or to state that every person has at most one father and one mother. It furthermore embraces the 'anything goes' mentality of RDF, which leads to conceptual, computational and decidability issues.

3.4 Requirements for Web-Based Knowledge Representation

The ideas behind the Semantic Web initiative was met with a fair amount of enthusiasm by the knowledge representation and acquisition community. The ideal of knowledge sharing that was so prominent in the literature of the early nineties (Neches et al., 1991; Gruber, 1993; Breuker and Van De Velde, 1994, etc.) was not feasible when confined to the traditional libraries or knowledge servers, but the web, and the semantic web, seemed its perfect partner. Knowledge sharing need no longer be controlled through publication in a library, but can be *unattended* and made possible simply by the availability of a knowledge representation at some URL. One of the first proposals for a web-based ontology language – coinciding with the development of RDF and RDFS – was the

[23] See Section 2.5.1 for a description of model theory.

[24] Although some people may turn into vegetables.

Simple HTML Ontology Extensions language, SHOE for short (Heflin et al., 1999)). SHOE allowed the semantic annotation of webpages, had an XML syntax and was frame-based, i.e. concepts are defined as frames with slots (cf. Section 2.2.4). However, like RDF Schema, it did not have a well-defined semantics.

An important requirement was that a web-based knowledge representation language should sanction standard inference. Reasoners complying with the standard should produce the same inferences, given the same input. This is a significant difference with highly expressive languages such as the Knowledge Interchange Format (Genesereth and Fikes, 1992, KIF), which itself did not have reasoners but instead relied on a translation to less expressive languages for which implementations were available. The web ontology language was to be more than just a *specification* language that can be used to exchange formal theories between systems, but rather a *representation* language in its own right. This requirement gains an extra edge as knowledge published on the web is not merely intended for human consumption, but also, and more importantly, for direct communication between different systems that act as knowledge components in a larger, distributed knowledge based system. Only systems that commit to the same representation will be able to properly interpret each others messages.

Information exchange between web-based systems becomes increasingly more mission critical: knowledge-based reasoning should not only be suitably efficient, but sound and complete as well (see Section 2.5.1). A web-based knowledge representation language is therefore subject to the *restricted language thesis* of Levesque and Brachman (1987) and needs to trade off expressive power with computational efficiency. The relaxed view of Doyle and Patil (1991) may well hold for applications where the user is in direct control, and understands the rationale of the system. But it is untenable on the web, where any user may use any knowledge representation for any purpose in any system. In other words, because standard reasoning should be efficient and decidable, the language should be limited in computational complexity and expressiveness.

From the outset, it was clear that the language should strike a fine balance between the open nature and expansiveness of the web on the one hand, and formal, well-defined semantics on the other. It should therefore build on top of existing (semantic) web standards such as XML, RDF, and RDF Schema (cf. Figure 3.1), and allow knowledge engineers and users to freely extend and reuse the knowledge made available on the web. This meant not only that ontology files should be able to *import* each other from URL's, but also that the semantics of every ontology should take into account that it can be *imported* and extended. In other words, the ontology language should assume that any ontology represents incomplete knowledge and adopt the *open world assumption*; a reasoner should only make inferences based on those statements that are known to be true. This is in stark contrast with common practice in many rule based systems, where inference generally takes place on the assumption that any statement not known to be true is false. In a closed system this closed world assumption (or negation as failure principle) is quite sensible, but it can be quite dangerous on the web. For instance, it would mean that what is inferred for an individual given the axioms in an ontology A may be inconsistent with that which is inferred for the same individual when imported to another

ontology B. In the former case, any inference based on the axioms in B is assumed not to hold, where they do hold in the latter case.

The DAML+OIL language of Connolly et al. (2001)[25] was a proposal for a semantic web knowledge representation language in the tradition of the earlier KRSS (Patel-Schneider and Swartout, 1993, see Section 4.2). It had a DL-style semantics and was a combination of two languages; the DARPA Agent Markup Language (DAML-ONT),[26] and the Ontology Inference Layer (Fensel et al., 2000, OIL).[27]

Both approaches were very similar, they were:

- languages with formal semantics,
- extensions of XML and RDF(S),
- intended for defining concepts and relations,
- to be shared on the *web*

DAML+OIL was the first knowledge representation language that combined the principle of knowledge sharing with the formal semantics and limited expressiveness of DL, allowing for sound and complete automated reasoning (Baader et al., 2003). Where the semantics of OIL relied on a translation to the description logic $\mathcal{SHIQ}(d)$ (Horrocks, 2000) and was not necessarily compatible with RDF, DAML+OIL was rather a syntactic variant of a description logic and aimed to maximise compatibility with RDFS. It was the first proper knowledge representation language specifically tailored for the web.

3.5 The Web Ontology Language

OWL extends the syntax and semantics of RDFS with a fair number of constructs and combines the description logics semantics of DAML+OIL with more rigorous semantic and syntactic compatibility regarding RDF and RDFS. It furthermore prescribes how ontologies should be published on the web, and incorporates a mechanism akin to that of SHOE for ontology imports. In the following I give a brief, non exhaustive, overview of the OWL vocabulary and its semantics. The interested reader may refer to Horrocks et al. (2003); Bechhofer et al. (2004) for a full overview.

3.5.1 OWL

The various requirements for a web ontology language led to the specification of three *species*, each emphasising different language features: OWL *Full*, *DL*, and *Lite*. OWL Full is a semantic extension of RDFS that adds a number of knowledge representation primitives that were previously unavailable. It emphasises expressiveness and compatibility with RDFS: the RDFS interpretation of an OWL Full ontology is consistent with its OWL Full entailments; and any

[25] See `http://www.w3.org/TR/daml+oil-reference`

[26] Research funded by *Darpa*, the Defense Advanced Research Projects Agency, see `http://www.darpa.gov` and `http://www.daml.org`

[27] Developed within the EU funded OnToKnowledge project, see `http://www.ontoknowledge.org/oil`

OWL Full ontology is valid RDFS and vice versa. This means, amongst others, that in OWL Full the primitives of the language (i.e. the meta logical symbols) are part of the *domain*. However, as proved by Motik (2007), this mixing of logical and metalogical symbols in OWL Full leads to undecidability (see Section 3.3.2).

The second species, OWL DL, is a syntactic subset of OWL Full, but limits the semantics to the $\mathcal{SHOIN}(D)$ description logic (see Table 2.4). At the time this language was the maximally expressive, decidable description logic for which efficient algorithms were known and use cases existed. To retain decidability, the compatibility with RDFS is restricted to a specific subset of that language. This means that although every OWL DL ontology is valid OWL Full and thus RDFS, this does not hold the other way around. However, because any conclusion given the OWL DL semantics is a valid conclusion in OWL Full, the RDFS interpretation of an OWL DL ontology is consistent with its DL entailments.

Because $\mathcal{SHOIN}(D)$ is very expressive, it allows the construction of exceedingly complex expressions which can be quite hard to grasp for an ontology engineer. Furthermore, reasoning with this logic is computationally expensive and intractable. The third species, OWL Lite, is meant to alleviate some of the problems of its bigger brother. It is a syntactic subset of OWL DL, where the semantics is designed to be limited to the $\mathcal{SHIF}(D)$ description logic. Similar to OWL DL, any OWL Lite conclusion is a valid OWL DL conclusion, and the RDFS interpretation of an OWL Lite ontology is therefore consistent with its DL entailments. Unfortunately it soon turned out that the syntactic limitations imposed on OWL Lite did not, in fact, limit it to $\mathcal{SHIF}(D)$, and most of the semantics of OWL DL can be expressed using elaborate combinations of OWL Lite constructs.

OWL follows the distinctions of DL between *class* axioms, *property* axioms and *individual* assertions. Figure 2.11 shows how the OWL DL constructs owl:Class and owl:Individual map onto their corresponding DL constructs.

Class Axioms Since not all RDFS classes are valid OWL DL and OWL Lite classes, OWL introduces the owl:Class which is the rdfs:subClassOf rdfs:Class consisting of all valid OWL classes. Because in OWL Full all RDFS classes are valid, owl:Class is equivalent to rdfs:Class under OWL Full semantics.

Just as in RDFS, owl:Classes can be related using rdfs:subClassOf properties. All OWL classes are a subclass of the class owl:Thing – which in OWL DL is equivalent to $\top$. The class owl:Nothing represents the empty class – in OWL DL equivalent to $\bot$ – and is defined as the complement of :owl:Thing. In OWL Full owl:Thing is equivalent to rdfs:Resource.

Two classes that are the rdfs:subClassOf each other, are *equivalent*, which can be directly expressed using the owl:equivalentClass property. An owl:Class is either *named* or *anonymous*. A named class is defined using *class axioms* that restrict it to be disjoint with, equivalent to or the subclass of one or more other classes. Disjointness of two classes means that the intersection of both is the empty set. There are three types of anonymous classes:

- *Nominals* are defined by exhaustively enumerating all individual class members, for instance:[28]

[28]The _:x in the example is a *blank node*, an RDF resource without a URI. This blank node can

```
:Animal a owl:Class ;
    owl:equivalentClass _:x ;

_:x owl:oneOf (:fluffy :felix :bertha :snuffy :max :babel) .
```

states that the class :Animal is equivalent to the set consisting of the individuals :fluffy, :felix, :bertha, :snuffy, :max and :babel. In DL-style syntax:

$$\text{Animal} \equiv \{\mathit{fluffy}, \mathit{felix}, \mathit{bertha}, \mathit{snuffy}, \mathit{max}, \mathit{babel}\}$$

- *Operator* classes are defined using the standard set theoretic operators union, intersection and complement, for instance:

```
:Animal a owl:Class ;
  owl:equivalentClass [
    owl:unionOf (:Mammal :Fish :Reptile :Insect)] .
```

states that the class :Animal is equivalent to the union of the classes :Mammal, :Fish, :Reptile and :Insect. In DL-style syntax:

$$\text{Animal} \equiv \text{Mammal} \sqcup \text{Fish} \sqcup \text{Reptile} \sqcup \text{Insect}$$

- *Restriction* classes are defined as the set of all individuals that satisfy a restriction on a property. The restriction can be *existential* or *universal*, i.e. it expresses that some or all values of a property at the restricted class must be a member of some other (anonymous) class. Or, it may restrict the *cardinality* or prescribe a specific individual *value* of the property. For instance:

```
:Animal a owl:Class ;
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :cell_type ;
    owl:allValuesFrom :Eukaryote ] ;

  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :cell_type ;
    owl:minCardinality 1 ] ;

  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :friend ;
    owl:hasValue :joe ] .
```

states that the class :Animal is the subclass of all things that have only Eukaryotic cells, have at least one value for the cell type property, and are the individual :joe's friend. Multiple rdfs:subClassOf statements on one

be left implicit in the Turtle syntax by placing the triples in which the blank node occurs between square brackets (see the operator class example). The space separated list of resources between parentheses is the Turtle abbreviation for an rdf:List.

class are, as in RDFS, interpreted as the *intersection* of the super classes. In DL-style syntax:[29]

$$\begin{aligned} \mathsf{Animal} &\sqsubseteq \mathsf{cellType}\ \textbf{only}\ \mathsf{Eukaryote} \\ &\sqsubseteq \mathsf{cellType}\ \textbf{min}\ 1 \\ &\sqsubseteq \mathsf{friend}\ \textbf{value}\ \mathit{joe} \end{aligned}$$

What can be confusing is that OWL allows one to state many things in different, but equivalent ways. For instance, note that the owl:minCardinality restriction in the definition of Animal is equivalent to:

$$\mathsf{Animal} \sqsubseteq \mathsf{cellType}\ \textbf{some}\ \mathsf{owl{:}Thing}$$

Also, the owl:allValuesFrom restriction is trivially satisfied if an individual has no value for the cell_type property.

Individuals Individuals can be said to belong to a class in the usual RDFS way, i.e. by asserting an rdf:type triple between the individual and class. Asserting property values on individuals is similarly straightforward, e.g. to state that :joe is a :friend of :babel the fish, we write:

```
:babel a :Fish ;
    :friend :joe .
```

Or, in DL syntax:[30]

$$\begin{aligned} \mathit{babel} &\in \mathsf{Fish} \\ \mathsf{friend}(\mathit{babel}, \mathit{joe}) & \end{aligned}$$

As mentioned earlier, individuals in OWL are not subject to the unique name assumption. We have seen that two individual names can be inferred to have the same model through the use of a functional property. The same can also be achieved in a less round-about way by asserting an owl:sameAs relation between two individuals. The other way around, the owl:differentFrom property can be used to assert that two individuals do not have the same model.

Remember that in description logics individuals are elements and classes are subsets of the domain $\Delta^{\mathcal{I}}$. As a consequence, OWL DL only allows individuals in the subject and object position of property relations, because otherwise subsets of the domain are themselves elements in the domain; which leads to undecidability (Motik, 2007).

Property Axioms Another important extension of the RDFS vocabulary is the addition of property axioms beyond the rdfs:domain and rdfs:range attributes:

[29] The examples in this chapter and chapters 5,7 do not use the standard DL-style syntax, but a more readable form that is akin to the syntax used by common OWL editors such as Protégé and TopBraid Composer.

[30] Since the RDF syntax for OWL is quite verbose – even when using Turtle – the following uses the DL-style notation for OWL constructs.

- Any property can be stated to be the owl:inverseOf another property. Any two individuals related via a property in one direction, are related by the inverse of that property in the other direction. As a consequence, the domain of a property is equivalent to the range of its inverse and vice versa.

- An OWL property can be stated to be the rdfs:subPropertyOf another property. The set of models of that property is a subset of the models of its super property. Consequently, the domain and range of the property are the respective subclasses of the domain and range of the super property.

- *Functional* properties have exactly one individual as range; this means that any individuals asserted in the object position of that property have the same model, and are thus *the same*. This construct should not be confused with cardinality restrictions on classes. A functional property expresses a *global* cardinality restriction; but it applies separately to every individual in the knowledge base on which the property is used, regardless of the class it belongs to. For instance, consider the functional father property. Then from

$$\begin{aligned} \mathsf{father}(\mathit{babel}, \mathit{bubba}) \\ \mathsf{father}(\mathit{babel}, \mathit{elvis}) \end{aligned}$$

$$\begin{aligned} \mathsf{father}(\mathit{max}, \mathit{ludwig}) \\ \mathsf{father}(\mathit{max}, \mathit{beethoven}) \end{aligned}$$

  an OWL reasoner will infer that $\mathit{bubba} = \mathit{elvis}$ and $\mathit{ludwig} = \mathit{beethoven}$.

- *Inverse functional* properties express a similar global cardinality restriction but have exactly one individual as *domain*; any individuals asserted in the subject position, for the same object of that property have the same model:

$$\begin{aligned} \mathsf{father\_of}(\mathit{bubba}, \mathit{babel}) \\ \mathsf{father\_of}(\mathit{elvis}, \mathit{babel}) \end{aligned}$$

  gives $\mathit{bubba} = \mathit{elvis}$. An inverse functional property has the same effect as the inverse property of a functional property, but is more concise as it does not require the existence of a functional property.

- *Transitive* properties allow property values to propagate along a chain of connected properties. Examples of transitive properties are taxonomic relations such as the rdfs:subClassOf property and mereological relations such as part_of, e.g.:

$$\begin{aligned} \mathsf{part\_of}(\mathit{piston}, \mathit{engine}) \\ \mathsf{part\_of}(\mathit{engine}, \mathit{car}) \end{aligned}$$

  gives $\mathsf{part\_of}(\mathit{piston}, \mathit{car})$.

- *Symmetric* properties express that any two individuals related via the property in one direction are also related in the converse direction. In other words, a symmetric property is equivalent to its inverse. Given, $\mathsf{sibling}(\textit{eefje}, \textit{suzan})$ the reasoner infers $\mathsf{sibling}(\textit{suzan}, \textit{eefje})$.
- Any two or more properties can be stated to be equivalent to each other. The classes in the domain and range of the properties are correspondingly equivalent, and any pair of individuals related via one property is also related via the other.

The semantics of these property axioms is in fact very powerful because the axioms hold *globally*. A second reason is that where (in DL) class axioms only effect the semantics in the TBox, property axioms in the RBox can have a significant implications for axioms in the the TBox, such as the effects on classes in the domain and range of inverse, equivalent and sub properties.

The domain and range of properties in OWL are in principle not restricted in any way, i.e. properties and restrictions can range over concrete data values such as strings and integers. However, it has been shown that the combination of datatypes and description logics, i.e. the incorporation of a concrete domain into a concept language, may cause undecidability (Lutz, 1999). To retain decidability, OWL DL adopts the solution of Horrocks and Sattler (2001); Baader and Hanschke (1991) who introduce an additional domain for concrete data values $\Delta_D^{\mathcal{I}}$ in N_{I_c} which is disjoint with the domain of abstract individual objects $\Delta^{\mathcal{I}}$ in N_{I_a}, where $N_I = N_{I_c} \cup N_{I_a}$. Analogous to classes and individuals in $\Delta^{\mathcal{I}}$, datatypes are interpreted as subsets of $\Delta_D^{\mathcal{I}}$ and data values are elements of $\Delta_D^{\mathcal{I}}$. In practice, this resulted in a distinction between two disjoint types of properties: properties that have only literal values, *datatype* properties or concrete roles in N_{R_c}, and properties that have only individuals as values, *object* properties or abstract roles in N_{R_a} where $N_R = N_{R_c} \cup N_{R_a}$. Formally, a datatype property D is defined as a binary relation, a subset of the set of all object data value pairs: $\mathsf{D}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$. In other words, properties can not have data values in the subject position in OWL DL. Consequently, datatype properties cannot have an inverse, be symmetric, inverse functional or transitive. This restriction does not hold in OWL Full as $\Delta_D^{\mathcal{I}}$ and $\Delta^{\mathcal{I}}$ are not disjoint under Full semantics.

Although generally a property can be of any or all of the property types described, some combinations between property types and class restrictions require extra care or are forbidden in DL (Horrocks et al., 2006). In particular *composite* properties may cause decidability issues in several cases (Horrocks et al., 1999). A composite property is any property whose semantics is defined as a (possible) sequence of (other) properties: i.e. the transitive properties. A *complex* property is a property whose definition is defined in terms of a composite property, i.e. the super properties, equivalent properties and inverse of transitive properties. In DL, complex properties are not allowed in cardinality restrictions, nor are they allowed to be functional or inverse functional.

3.5.2 OWL 2

In October 2007, the W3C started a new working group to develop a successor to the OWL Web Ontology Language: OWL 2. This language extends the original OWL with a number of features for which effective reasoning algorithms

are available, and which meet real user needs.[31] Until now, most research on OWL 2 has been directed towards a new version of the description logics dialect of OWL 1,[32] and is based on the $\mathcal{SROIQ}$ description logic described by Horrocks et al. (2006), but extends it with datatypes and punning (see below).

A large part of the development of OWL 2's features has taken place during, and in between OWLED workshops, i.e. outside of the W3C standardisation process.[33] At the start of the working group, efficient implementations of OWL 2 were already available in standard reasoners such as Pellet, FaCT++, Racer and KAON2.[34] Because of its early and wide adoption by both implementers and users, the OWL 2 effort has a good chance of reaching W3C recommendation status.

OWL 2 has a modular and extensible architecture in the form of language profiles, user defined datatypes and semantic annotations. It resolves several ambiguities in the RDF mapping, such as the ability to use owl:oneOf with or without an owl:equivalentClass axiom, and extends the OWL 1 vocabulary with shorthand notations for frequently used combinations of OWL 1 primitives such as disjoint union.

Some of OWL 2's most significant additions lie in the expressiveness for property axioms. *Asymmetric* properties can be used to express e.g. that if something is bigger than something else, this does not hold vice versa, in other words: the property is disjoint with its inverse. The asymmetry of a property can thus also be expressed in terms of two *disjoint properties*. Disjointness between two properties means that their sets of models are disjoint; no pair of individuals can be in both sets. This can be used to express e.g. the disjointness of the brotherhood and sisterhood properties: if Mary is Bob's sister, she is not his brother. A nice side-effect of property disjointness is thus that it can be used to state that some individual is *not* related to another individual via some property. OWL 2 provides syntactic sugar in the form of *negative property assertions* on individuals that can express this without the need to create a dummy disjoint property.

The feature of *property chain inclusions* – also called complex role inclusion – allows the transitive 'inheritance' of some property over a chain of properties Horrocks et al. (2006). For instance,

$$\mathsf{owns} \circ \mathsf{has\_part} \sqsubseteq \mathsf{owns}$$

expresses that if some individual owns an individual, it is also the owner of that individual's parts.[35] As shown in Chapter 7 role inclusions are very powerful primitives. Important to note, however, is that the chain of properties on the left is only a sub property of the property on the right hand side: they are not equivalent. Also, property chains, like transitive properties, are *compound*, which means that their use in combination with other property and class axioms is subject to the same restrictions under OWL 2 DL.

[31] See the WG website at http://www.w3.org/2007/OWL/. At the time of this writing, the exact specification of OWL 2 was not determined yet. Please refer to the website for the latest version.

[32] To avoid confusion, from here on I refer to the original OWL standard as OWL 1

[33] OWLED: "OWL: Experiences and Directions". See http://www.webont.org/owled/

[34] See http://pellet.owldl.com, http://owl.man.ac.uk/factplusplus/, http://www.racer-systems.com and http://kaon2.semanticweb.org/ for respective descriptions and downloads.

[35] Where $\circ$ is a concatenation operator.

OWL2 properties can be *reflexive* or *irreflexive*, to express such things as that every thing is part of itself, or that no effect can be its own cause. Global reflexivity of a property makes it apply to *all* individuals. A more subtle way to express reflexivity is *local reflexivity* of properties in class descriptions. This enables us to express that each member of the restricted class is related to itself via that property. A local reflexivity restriction takes the form of a '**self**' restriction on an object property. For instance, the class definition:

$$\mathsf{Narcissist} \equiv \mathsf{Person} \sqcap \mathsf{likes}\ \textbf{some self}$$

states that the class Narcissist is equivalent to the set of persons who like themselves. To indicate the difference, if the likes property were itself reflexive then *every* individual would like itself. The **self**restriction can only be used in combination with an existential restriction.

A second new class axiom is the *qualified cardinality restriction* (QCR). QCR's add the ability to restrict the cardinality of a property to a particular range, e.g. a table has exactly four legs as parts, but can have more parts which are not legs:

$$\mathsf{Table} \sqsubseteq \mathsf{has\_part}\ \textbf{min}\ 4\ \mathsf{Leg} \sqcap \mathsf{has\_part}\ \textbf{max}\ 4\ \mathsf{Leg}$$

Punning

As pointed out multiple times, description logics retain decidability by disallowing meta modelling and redefinition of meta logical symbols that are part of the vocabulary of the language itself. In practice, this means that the sets of names for the vocabulary, classes, properties and individuals are mutually disjoint (see Section 2.5.1). OWL 2 drops this requirement, but obtains the same effect by explicitly marking the proper interpretation for each occurrence of a name: names are given a *contextual semantics*, c.f. Motik (2007). This means that names can be treated as *any or all of* these types, e.g. the meaning of a name as class is in no way affected by the meaning of a name as individual. This trick is called *punning*, wordplay. Although the interpretation of a name may vary, the name is still considered to denote the same entity. Punning enables straightforward meta modelling without the impact on decidability that RDF and OWL Full modelling have. Because of interaction with the semantics of OWL Full, punning is not allowed between data and object properties.

Publishing and Metadata

The support for metadata and annotation of ontologies in OWL 1 was both under specified and limited. These limitations are largely the result from a lack of experience in how OWL ontologies are used by various users and systems. A few years onwards, this experience is present and allows OWL 2 to embody a more comprehensive view on annotations and metadata.

Firstly, OWL 1 supports the standard rdfs:label and rdfs:comment properties to respectively give meaningful names and attach descriptions to classes, properties and individuals. These properties are interpreted as owl:Annotation Property, a special type of property that does not carry any semantics in DL, but is interpreted as rdf:Property under OWL Full semantics. Users are free to define additional annotation properties, but their use is severely limited in DL.

Annotation properties cannot be used *in* class restrictions, e.g. to define the set of all persons with more than one rdfs:label name. And they cannot be used *on* class restrictions either. Especially this last restriction was considered an unnecessary shortcoming, and OWL 2 DL extends OWL with *axiom annotations*, that is any axiom (such as a class restriction) can be annotated.

Furthermore, OWL 2 extends the annotation system with rdfs:subPropertyOf relations, and the specification of domain and range for annotation properties. Under DL semantics these do not carry semantics, but the extension allows a larger portion of OWL Full ontologies to be valid in OWL 2 DL as well. Ways are currently being investigated to allow *rich* annotations which can be used to attach information to axioms that is interpretable by extensions of OWL reasoners.[36] For instance, PRONTO[37] extends the Pellet reasoner with a probabilistic reasoning engine that uses annotations on rdfs:subClassOf to assert the probability of that relation to hold between two classes.

Another area where OWL 1 was significantly underdeveloped is its means to resolve owl:import relations between ontologies. In OWL 1, the owl:import property is simply defined as a transitive property that holds between two ontologies, i.e. the domain and range of owl:import is owl:Ontology. However, because the specification is silent on *how* ontologies are to be identified on the web, there is no standard means by which the import mechanism can be used. This weak definition was a relatively late addition to the language, resulting from a compromise between the ontology-as-theory and ontology-as-document perspectives. Regardless of its ambiguous definition, it turned out that this owl:import property was widely used. The OWL 2 specification (Motik et al., 2009) is therefore much more explicit in this respect, and recommends a method that combines the imports mechanism with simple versioning. OWL 1 defines several ontology properties for versioning (owl:priorVersion, owl:backwardCompatibleWith and owl:incompatibleWith) but again, these do not have a prescribed normative, nor recommended, effect on the semantics of imports. Because there was no recipe for importing a particular version of an ontology, each OWL tool used its own particular way to deal with versions and perform resolution of ontology URI's to ontology documents.

In a nutshell, the mechanism in OWL 2 works as follows. If an OWL 2 ontology imports another ontology, this is done from the URL dereferenced (in the standard way) by the URI value of the owl:imports property. The imported ontology should have either that URI as the value of its ontology URI (i.e. the URI of the owl:Ontology element), or as the value of its owl:versionInfo property. Because user agents, such as ontology editing tools may specify their own method for resolving URI's to locations, the ontologies need not necessarily be published on the web, but may be accessed from ontology repositories or a local file system. If an ontology is imported by another ontology, then its import closure becomes part of the import closure of the importing ontology. The axiom closure of an ontology is the smallest set that contains all axioms from all ontologies in its import closure.[38] The versioning properties from OWL 1 may be used by user agents to raise a flag when e.g. the value of an owl:incompatibleWith property corresponds with a owl:versionInfo or onto-

[36] See `http://www.w3.org/2007/OWL/wiki/Annotation_System`

[37] See `http://clarkparsia.com/weblog/2007/09/27/introducing-pronto/`

[38] See also definitions 5.4.2 and 5.4.3 in Section 5.4 for a more formal discussion.

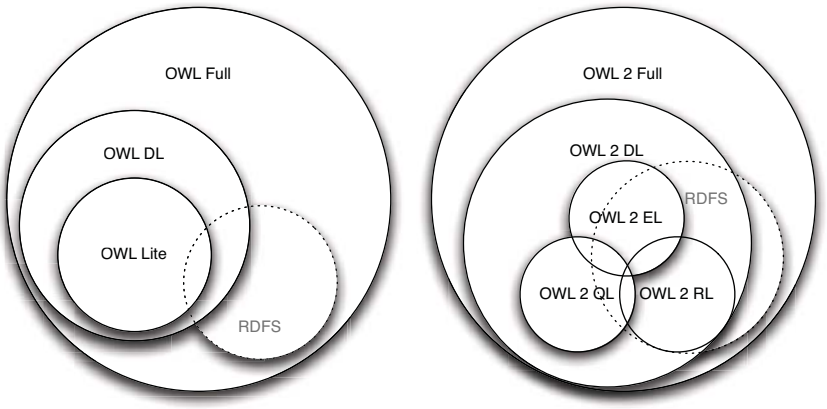

Figure 3.4: OWL 1 species vs. OWL 2 Profiles

logy URI.

Profiles

The OWL 2 specification includes a number of so-called *profiles*, some of these are well-known tractable subsets of the OWL 2 DL specification, others are more expressive, but do not have the full semantics of OWL 2 Full (Cuenca Grau et al., 2009). The motivation for providing these profiles is that many existing ontologies tend to use only a particular subset of the language constructs available in DL; and significant increase of reasoner performance can be achieved through reasoning using a less expressive language. It is thought that a standard library of logical profiles with a particularly likeable tradeoff between expressiveness and computational complexity can overcome some of the problems experienced when defining the OWL 1 Lite version. In particular the profiles are:

- restricted by *syntax*. The semantics of a profile's syntax is provided by the OWL 2 DL specification.
- defined by logics that can handle at least some interesting inference service in polynomial time with respect to either:
  - the number of facts in the ontology, or
  - the size of the ontology as a whole.

This section gives a brief overview of the profiles defined in OWL 2 and their typical application areas, see Figure 3.4.[39]

OWL 2 EL The EL profile, based on the $\mathcal{EL}++$ DL introduced in Baader et al. (2005), is an extension of the $\mathcal{EL}$ description logic. Its primary strength lies in

[39] For an up-to-date overview, see `http://www.w3.org/2007/OWL/wiki/Profiles`

the ability to reason in polytime on ontologies with large TBoxes, and was designed to cover the expressive power of several existing large-scale ontologies, such as

SNOMED-CT®
is a large-scale commercial ontology that defines the international standardised terminology of the IHTSDO,[40] which is used in the health care systems of both the US and the UK. It consists of about 500k named class definitions.[41]

Gene Ontology
is an ontology of 25 thousand terms related to genes and gene properties.[42]

GALEN
About 95% of this closely interlinked multi-lingual ontology of medical terms is covered by $\mathcal{EL}++$. The GALEN ontology contains about a thousand class definitions.[43]

Where $\mathcal{EL}$ supports *conjunction* and *existential* restrictions, $\mathcal{EL}++$ extends this with nominals and role inclusions. It is lightweight and supports sound and complete reasoning in polynomial time. The most significant difference with OWL 2 DL ($\mathcal{SROIQ}$) is that it drops the owl:allValuesFrom restriction, though it does support rdfs:range restrictions on properties, which can have a similar effect. $\mathcal{EL}++$ is not a profile of OWL 1 DL as it supports the complex role inclusion axioms of OWL 2, and it is more expressive than OWL 1 Lite in that it allows for existential owl:someValuesFrom where OWL 1 Lite doesn't.

OWL 2 QL Reasoners developed for OWL DL 1.0 and 1.1 are optimised for reasoning on TBox axioms, and are relatively inefficient when dealing with ontologies that have relatively uncomplicated class definitions, but contain a large number of ABox assertions. The QL profile of OWL 2 was developed to efficiently handle query answering on such ontologies, and adopts technologies form relational database management. It is based on the DL-Lite description logic of Calvanese et al. (2005). By itself DL-Lite is a very restricted a profile of both OWL 2 DL and OWL 1 DL but the QL fragment of OWL 2 extends DL-Lite with more expressive features such as the property inclusion axioms, i.e. owl:subPropertyOf, and functional and inverse-functional object properties of OWL 1. However, it also includes the top and bottom roles of OWL 2, which adds expressiveness beyond that of OWL 1 DL.

OWL 2 RL The OWL 2 RL profile is based on so-called Description Logic Programs (Grosof et al., 2003), which is a subset of OWL DL 1.0 and the Horn profile of First Order Logic (FOL) (see Figure 3.5). DLP enables the interaction

[40] International Health Terminology Standards Development Organisation, see `http://www.ihtsdo.org`

[41] Systematized Nomenclature of Medicine, Clinical Terms, see `http://www.snomed.org`

[42] See `http://www.geneontology.org/`

[43] Generalised Architecture for Languages, Encyclopaedias and Nomenclatures in medicine, see `http://www.openclinical.org/prj_galen.html`

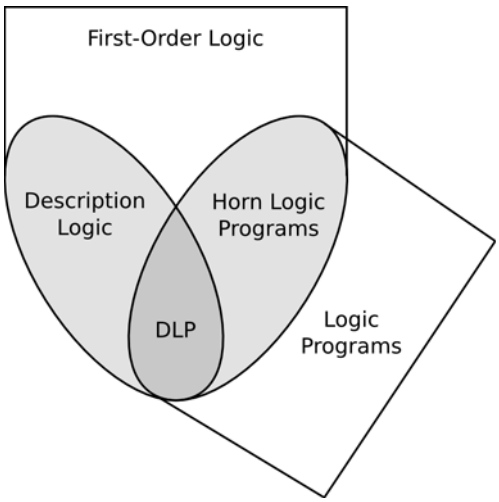

Figure 3.5: The DLP profile, adapted from Grosof et al. (2003)

between DL and rules[44], and effectively allows knowledge engineers to build rules 'on top' of ontologies and vice versa. DLP axioms can be translated to Horn clauses in a standard way. Nonetheless, the mapping is to some extent syntactic as OWL DL reasoners adopt monotonicity, and adopt the open world assumption, whereas logic programming engines use closed world reasoning and allow non-monotonicity.

The RL profile is a syntactic fragment of OWL 2 DL, but differs from the QL and EL profiles in that its semantics is partially given by a set of rules that extend the RDFS interpretation of valid RDF graphs of this profile. All entailments of OWL 2 DL reasoner over this fragment will be trivially sound and complete, but an OWL Full (or RDFS) reasoner will additionally have to implement the rules to ensure soundness. The OWL Full/RDFS interpretation is an extension of OWL 2 RL that was originally developed by Oracle[45] which, similar to QL, enables efficient OWL reasoning on top of a relational database. A forward-chaining reasoner for this profile has been implemented as part of Oracle 11g. The primary rationale behind the development of OWL 2 RL Full is that existing DL reasoners (Pellet, KAON2) cannot handle reasoning on the often quite substantial amount of entries in databases.

3.6 Discussion

The development of the Semantic Web has lead to a number of technologies that, in conjunction, constitute a balanced, layered approach for representing knowledge on the web (cf. Figure 3.1). Despite some initial problems due to its unclear and non-standard semantics, RDFS has proved to be a solid founda-

[44] As defined in RuleML, see `http://www.ruleml.org`, and now RIF, the Rule Interchange Format, `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`

[45] Also known as OWL Prime, or RDFS 3.0, see `http://www.oracle.com/technology/tech/semantic_technologies/pdf/semantic11g_dataint_twp.pdf`.

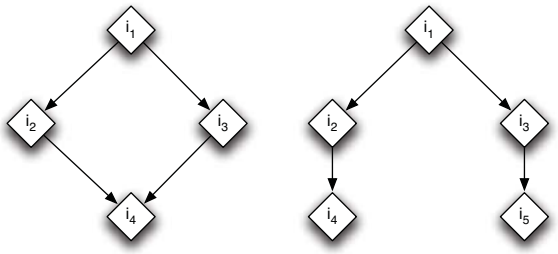

Figure 3.6: Diamond vs. tree model

tion for the now standard representation language OWL. The representation of knowledge on the web posed several requirements for this language in addition to those outlined in Chapter 2, with respect to both semantics and syntax. On the one hand, compatibility of its syntax and semantics with the lower layers in the cake, RDF and RDFS, had to be ensured. On the other hand, it should fit the tradition in knowledge representation, and description logics in particular. Inference on the axioms in ontologies expressed in OWL is *monotonic* to remain unaffected by the addition of new information. OWL therefore adopts the open world assumption. Lastly, efficient algorithms for the language have been shown to exist, i.e. it is *decidable*, and have been implemented in several reasoners.

The above aspects of the web ontology language are a consequence of the way it deals with several trade-offs (Horrocks et al., 2003). The most important of these is perhaps the one between *decidability* and *expressiveness*. In this respect, the development of OWL has been, and still is a cautious one: new features are only added to the language provided that some suitably efficient algorithm is known. Decidability of OWL, as a fragment of first order logic, was therefore only possible by sacrificing expressiveness.

The most prominent hiatus in expressiveness lies exactly where rule based formalisms find their strength. As most decidable description logics, $\mathcal{SROIQ}$ has the tree-model property, i.e. every concept in the logic has a model only if it has a *tree*-shaped model. In a tree-shaped model the interpretation of properties defines a tree shaped directed graph. In its simplest form, the problem is that class descriptions cannot be used to distinguish between the two patterns of individuals in Figure 3.6, i.e. a class description cannot enforce the existence of an owl:sameAs relation between individuals i_4 and i_5. In short this also means that defining the pattern of Figure 7.4 is theoretically impossible in this language.

Although several decidable fragments of logic programming languages such as Datalog exist that *can* express diamond-shaped models, these are extensions of first order logic which operate under the closed world assumption and thus do not meet the requirement of monotonicity. Furthermore, the development of web-based rule languages has proven to be slow going because of the enormous range of different rule-based languages on the market today. After its start in 2005, the Rule Interchange Format (RIF) working group has only released its first public working drafts in 2008 (including a specification

of its interoperability with OWL).[46]

In general it can be said that any knowledge representation language intended to be used for reasoning necessarily must find a balance between expressiveness and computational complexity: the trade-off is inevitable. In Chapter 7 I discuss the theoretical considerations and practical limitations of this trade-off in more detail. However, whether what *can* be expressed is sufficient or not depends on what *should* be represented.

Despite the occasional drop of the 'O' word, inevitable in discussing a web *ontology* language, this chapter steered clear of any of the pitfalls surrounding the term 'ontology'. Although OWL was presented as a web-based knowledge representation language, the fact that it is called an ontology language is not immediately obvious. The next chapters approach this confusion head-on. Chapter 4 elucidates the different interpretations of the term (and there are a couple), leading to the idea of an ontology as knowledge representation artefact that can be *built*. Nonetheless, an ontology remains a special breed, which becomes evident in the discussion of the issues surrounding ontology construction in Chapter 5.

[46] See `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`.