An assessment of RDF/OWL modelling

Dave Reynolds, Carol Thompson[1], Jishnu Mukerji[1], Derek Coleman[1]
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2005-189
October 28, 2005*

semantic web, modelling

The Semantic Web initiative offers a set of standards (RDF, RDFS and OWL) for the representation and exchange of information. We address the question of what sorts of modelling problems these standards can be most fruitfully applied to. What benefits can be gained from using them? What are the costs and trade-offs involved? We use some simplified examples drawn from work exploring the use of RDF/OWL within a systems management setting but the overall discussion should be relevant to a broad range of potential RDF/OWL applications. We do not assume in depth knowledge of the RDF and OWL standards and include a summary of their main features in order to make the discussion accessible. We identify the primary strengths of RDF/OWL as:

- support for information integration and reuse of shared vocabularies
- handling of semi-structured data
- separation of syntax from data modelling
- web embedding
- extensibility and resilience to change
- support for inference and classification, based on a formal semantics
- representation flexibility, especially ability to model graph structures
- ability to represent instance and class information in the same formalism and hence combine them

Weaknesses noted are:

- weak ability to validate documents
- expressivity limitations, particularly in terms of correlating across different properties of a resource
- performance
- XML serialization issues and impedance mismatch with XML tooling
- lack of familiarity and potentially high learning curve
- inability to natively represent uncertain data and continuous domains
- no built-in representation of processes and change

We conclude that RDF/OWL is particularly suited to modelling applications which involve distributed information problems such as integration of data from multiple sources, publication of shared vocabularies to enable interoperability and development of resilient networks of systems which can cope with changes to the data models. It has less to offer in closed world or point-to-point processing problems where the data models are stable and the data is not to be made available to other clients. It is also largely unsuited to domains involving continuous or fuzzy categories.

* Internal Accession Date Only
[1]HP SGBU – Software Global Business Unit

Approved for External Publication

An assessment of RDF/OWL modelling

Dave Reynolds, Carol Thompson, Jishnu Mukerji, Derek Coleman
2005-09-05

1 Summary

The Semantic Web initiative offers a set of standards (RDF, RDFS and OWL) for the representation and exchange of information. We address the question of what sorts of modelling problems these standards can be most fruitfully applied to. What benefits can be gained from using them? What are the costs and trade-offs involved?

We use some simplified examples drawn from work exploring the use of RDF/OWL within a systems management setting but the overall discussion should be relevant to a broad range of potential RDF/OWL applications. We do not assume in depth knowledge of the RDF and OWL standards and include a summary of their main features in order to make the discussion accessible.

We identify the primary strengths of RDF/OWL as:

- support for information integration and reuse of shared vocabularies
- handling of semi-structured data
- separation of syntax from data modelling
- web embedding
- extensibility and resilience to change
- support for inference and classification, based on a formal semantics
- representation flexibility, especially ability to model graph structures
- ability to represent instance and class information in the same formalism and hence combine them

Weaknesses noted are:

- weak ability to validate documents
- expressivity limitations, particularly in terms of correlating across different properties of a resource
- performance
- XML serialization issues and impedance mismatch with XML tooling
- lack of familiarity and potentially high learning curve
- inability to natively represent uncertain data and continuous domains
- no built-in representation of processes and change

We conclude that RDF/OWL is particularly suited to modelling applications which involve distributed information problems such as integration of data from multiple sources, publication of shared vocabularies to enable interoperability and development of resilient networks of systems which can cope with changes to the data models.

It has less to offer in closed world or point-to-point processing problems where the data models are stable and the data is not to be made available to other clients. It is also largely unsuited to domains involving continuous or fuzzy categories.

2 Table of Contents

3 What are RDF and OWL?

Before discussing the benefits and trade-offs involved in applying the semantic web technologies we first give a brief summary of what those technologies are. Readers familiar with the area may wish to skip this section.

The Semantic Web is a W3C initiative aimed at providing a common framework for data sharing and reuse across application, enterprise and community boundaries. The first foundation standards were published in 1999 but in recent years these have been revised and extended to the point where they are ready for serious application. HP has played a leading role in the development, application and tool support [20] for these standards.

Whilst the Semantic Web itself is aimed at large (web-scale) distributed information sharing, the standards and technologies are applicable within more constrained settings such as enterprise information systems.

At present the Semantic Web standards comprise two primary layers.

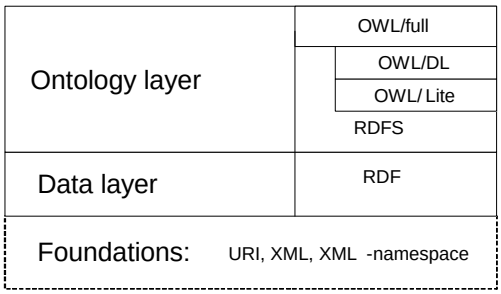


Fig 1. Core semantic web standards

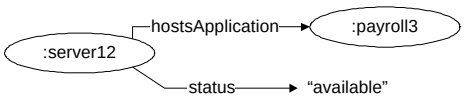
The first layer, the data layer, comprises the RDF (Resource Description Framework [1]) language which provides a common data representation that can be used for exchange and integration of machine-readable information.

The second layer, the ontology layer, supports the specification of the vocabularies to be used in RDF data. This ontology layer comprises the simple RDF Schema language (RDFS) and a richer Web Ontology Language OWL.[1] The full OWL language is a superset of RDFS so we will typically refer to the entire standards set from RDF up to OWL/full as RDF/OWL. In fact there are several restricted language profiles of OWL (OWL/Lite, OWL/DL) that provide different expressivity/performance trade-offs but these nuances will not be critical to much of the discussion.

3.1 RDF: The data layer

RDF represents information by means of atomic, logical statements. These statements take the form of *triples*: subject predicate object. For example[2],

```
urn:x-hp:server12   sdc:hostsApplication   bp:payroll3 .
urn:x-hp:server12   sdc:status             “available” .
```



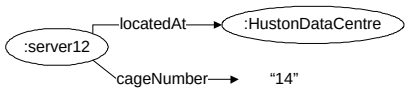
[1] The name OWL for the Web Ontology Language is in part justified by Winnie the Pooh in which Owl “could spell his own name WOL, and he could spell Tuesday so that you knew it wasn't Wednesday...".

[2] RDF statements can be written down using different syntaxes. For the examples we are using an informal syntax with one triple per line, together with a graphical representation of the same data. The URI references are abbreviated, so values such as “sdc:status” are intended to mean “the URI for the status predicate drawn from the sdc vocabulary” which in practice also means “the URI found by taking the namespace associated with the prefix ‘sdc’ and concatenating the localname ‘status’”.

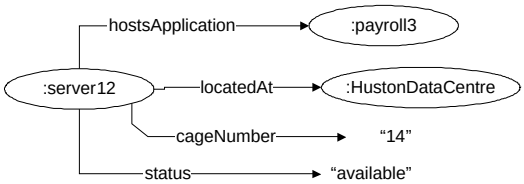
The subject (thing or resource being talked about) and the predicate (the property or relation being described) are identified by URIs. The object (the value of the predicate) may be either another resource or a literal value (a string or a typed value such as a number or date). The use of URIs is a key feature of the design. It provides a means for global naming (avoiding accidental clashes), it connects the statements to web resources (for example the thing described might be a web page or service with an identifying URL) and it provides a mechanism for publication and discovery (the description of the predicate is often published at the URI used to identify it).

The approach of representing information as sets of separate atomic statements is quite different from other modelling approaches such as XML or object oriented designs where values are grouped into packages (documents, or objects) with local structure (hierarchical tags, field names). It stems from the aims of the Semantic Web to support distributed sharing and integration of information. Two sets of RDF statements can be integrated by simply forming the union of the sets. For example, a separate asset database maintained by a separate application might supply the statements:

```
urn:x-hp:server12  asset:locatedAt    urn:x-hp:HoustonDataCentre .
urn:x-hp:server12  asset:cageNumber   14 .
```



These two sets of statements can be trivially merged, allowing us, for example, to infer that a problem with cage 14 at Houston will affect the payroll processing.



In contrast if two XML documents contain complementary information then any merging is application specific – you can't in general tell how the different syntactic elements in the two documents relate to one another.

RDF also provides the notion that resources have a type (called a *Class* in RDF). For example:

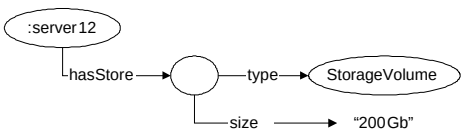
```
urn:x-hp:server12    rdf:type       ont:BladeServer .
```

However, in basic RDF these types are just opaque symbols (URIs) and it is the vocabulary or ontology layer that provides tools for specifying the semantics of these types.

There are a few other features of the core RDF data model which are worth mentioning but are not really critical to the later discussion.

First, one sometimes needs to make RDF statements about something that has no reasonable identifying URI or for which the exact identity is not relevant. In that case RDF provides the notion of a "blank node". This plays the role of an "existential variable" in logic languages, thus the statements[3]:

```
urn:x-hp:server12  sdc:hasStore    _:1 .
_:1                sdc:size        "200Gb" .
_:1                rdf:type        sdc:StorageVolume .
```



[3] Here we used the notation _:n for a blank node where the *n* is a purely local label to allow us to write the graph down, it is not an identity for the node.

tell us that there exists a 200Gb storage volume associated with the server but doesn't identify the specific storage volume involved.

Second, we sometimes want to make statements about the RDF statements themselves. For example, to say where a statement originated or how it was derived. To handle such cases RDF provides a mechanism[4] to assign URIs to statements and those URI labels can then be used as the subject (or indeed object) or other statements.

The RDF specifications include a standard XML serialization format. However, RDF is the data model rather than the serialization so other serializations are also possible and the data can be merged and processed independently of the serialization format chosen.

3.2 The vocabulary or ontology layer

So RDF provides a standardized means for exchanging data in a way which supports distribution and integration of information from multiple sources. However, that data is only useful for machine processing if the applications share or can exchange some common model of the domain. In concrete terms this means we need some way to specify the vocabulary elements that can be used and how they relate to each other. This is the job of the ontology layer.

The term *ontology,* in this context, just means a description of the concepts and relationships that make up a model of a domain. Using the term ontology implies that this description is formal, with well-defined semantics. We'll return later to the question of how ontologies differ from things such as schemas. For more details on the background to the term (as used in computer science rather than philosophy) see for example [2].

The base language for describing RDF ontologies is RDFS [3][5]. It provides facilities for defining class hierarchies, property hierarchies and domain and range declarations of properties. For example using RDFS we can state that a Class ont:Server is a super class of the earlier ont:BladeServer. Then an application that was querying for instances of ont:Server would find our server12 even if it knew nothing about the narrower class "blade server". In RDFS (and hence in OWL) we think of classes as representing sets of instances. They are categories into which we can place objects in the world, they don't represent data structures.

Similarly properties are just relations between resources and are global first-class things (this is rather different from the object-oriented notions of attributes as local names for parts the object data structure). We can relate properties to classes by defining the domain and range of a property. For example, we can declare that our property sdc:hostsApplication has domain ont:Server and range ont:Application. That would mean that we can infer that bp:payroll3 must be an ont:Application (since it's the object of an sdc:hostsApplication statement) even though we haven't yet explicitly stated that.

Just as RDF's data model is a set of logical statements or assertions about the world, the RDFS definitions are logical axioms about the classes and properties in the domain. The formal definitions mean these logical axioms can be used to infer new information (such as the inferences that server12 is an ont:Server and that bp:payroll3 is an ont:Application in our above examples). This is rather different from a schema language in which the aim is validation rather than inference.

OWL/full is a substantial extension of RDFS that enriches the set of axioms that can be stated about the classes and properties in our domain model. Amongst the important groups of additional capabilities are tools for declaring:

---

[4] Called *reification* in the jargon.

[5] RDFS stands for RDF Schema. This is a rather unfortunate name given that RDFS vocabularies are rather different from schemas in the sense of syntactic schemas such as DTDs or XML Schemas.

- features of properties (such as that one property is the inverse of another, or a property is transitive or symmetric, or that a property is functional[6] or inverse functional[7]);
- restrictions on classes - cardinality restrictions (such as all members of C have at least one value for property P), local range restrictions and value restrictions;
- class relationships, such as class C is the union of classes A and B;
- equality relationships - that two resources, classes or properties are the same, that two resources are different or that two classes are disjoint (have no members in common).

As well as defining necessary conditions for a class, OWL can be used to define sufficient conditions so that instances can be classified as being members of a class based on their properties. For example, we could define a notion of a GatewayServer as any Server that has two or more NetworkInterfaces and runs a ServerClass operating system and then query for any resource which meets that definition of the class GatewayServer.

For more details on the OWL language see [4].

One feature of the RDFS and OWL languages is that they are themselves just RDF vocabularies. For instance, our earlier example that ont:Server is a super class of ont:BladeServer is actually stated by means of the single RDF triple:

```
ont:BladeServer   rdfs:subClassOf    ont:Server .
```

This means that a single RDF document can contain both instance data and vocabulary extensions. Furthermore since everything (classes, properties, individual instances) is identified by global URIs then one ontology can easily refer to, build upon and combine terms used in other ontologies.

RDFS and OWL/full are very free in the way these language constructs can be used. For example, the same resource can be treated as both a class and an instance without problems. Whilst this has some advantages it can significantly complicate implementations and the OWL/full language is formally undecideable (which means that not all true statements can be inferred). The OWL/DL[8] sublanguage imposes some constraints on the way the OWL/full vocabulary can be used to reduce this complexity – in particular it requires that classes, properties, instances all be separate and that each property can only be used to give either literal values (owl:DatatypeProperty) or resource values (owl:ObjectProperty) but not both. The OWL/Lite sub-language further restricts OWL/DL to achieve lower implementation cost[9].

In this summary we've described OWL as a means to define the vocabulary to be used in RDF data. In fact an OWL ontology can be a useful artefact in its own right. Many groups who publish serious formal ontologies are shifting to the use of OWL for that alone (for example, the Gene Ontology[5] and the NCI Cancer Ontology[6]).

---

[6] Can have only one value.
[7] The inverse of the property is functional.
[8] DL stands for Description Logic. This is the name for a set of logics which are all subsets of first order predicate calculus. They are well suited for describing the features of classes of objects (hence the name) and trade-off expressive power in return for efficient implementations. The OWL language is based upon an expressive type of description logic.
[9] Inference is OWL/DL is decideable but has complexity class NEXPTIME whereas OWL/Lite is EXPTIME. Whilst these might sound like horrible complexity classes, in practice tractable algorithms exist for OWL/Lite and large parts of OWL/DL though ontologies that use all the features of OWL/DL simultaneously can be problematic.

4 The modelling approach and underlying assumptions

In this section we will expand on some of the nuances of these languages and what assumptions they are based on. In later sections we will then be able to explore the implications of these assumptions.

4.1 The Logical modelling approach

To recap, RDF/OWL models the world in terms of:

Resource instances – a resource is just a thing identified by a URI, it is not a data structure. It may represent some concrete item or an abstract concept.

Classes – a class represents a concept or category, it is just a set of resources and a resource can be a member of many classes. A class in OWL is not a computational object; it doesn't have methods that do things. It is more like a label.

Properties – a property is a relation between some resources and some values (which may be other resources or literals). It is like a two-column table in an object-relational model where the first column must be an objectID (the resource's URI).

Triples – a triple is a single statement relating one resource to one value via one property. RDF data is simply a set of such statements. Because the object of a triple may be another resource the set of triples can be thought of as forming a graph or network of relations between resource nodes.

An OWL *ontology*, is a set of classes and a set of properties that are used to describe information in some domain, together with a set of declarations (equations, axioms) which describe ways the classes and properties are related. For example saying that all resources that are member of class C have at least one value for property P.

The RDF/OWL languages have a well-defined formal semantics[10]. This means that given a set of statements in RDF/OWL it is possible to unambiguously conclude (infer) new additional statements that follow from the formal semantics. There is, of course, no absolute truth here - an OWL ontology is still just a model of world – nevertheless the formal semantics does make it possible to build up new vocabulary by reference to existing vocabulary.

There is one fundamental assumption here – that a logical, symbolic language is an adequate way to describe the domain being modelled. In some domains the categories are not discrete things but fuzzy[11] or continuous and need a different modelling approach from the logical "r is a member C" statements that can be made in RDF/OWL. To take a simple example, consider the notion of "a tall person". In OWL we can define a class TallPerson, as a subclass of Person, and can represent the actual height of a person but an individual is either a member of TallPerson or is not. Whereas in reality there is no universal height threshold which separates tall people from other people. Other modelling approaches exist (for example fuzzy logic, connectionist representations, various belief networks) that would allow us to capture the degree of membership of each individual in the TallPerson category.

[10] The semantics is defined in terms of a *model theory*. This is a standard way to define the semantics of a formal logic language dating back to Tarski.

[11] We are using the term *fuzzy* in a non-technical sense here, though is true that fuzzy logic is one approach to addressing some of the issues of representing continuous domains.

4.2 Identity and unique names

In RDF, resources, properties and classes are identified by URIs[12]. Since many things do not have natural existing URIs it is quite possible for different groups to invent different URIs to refer to essentially the same thing or concept. For this reason RDF and OWL do not make a *unique name assumption*. Just because two resources have different URIs it does not immediately follow that they are different.

OWL provides vocabulary to allow you to explicitly state that two resources are different or the same and there are several OWL constructs from which you can infer that two instances are the same. For example, we might define a property hasIPAddress as being an owl:InverseFunctionalProperty so if you find two resources that have the same IPAddress then you can infer that they are actually different labels for the same thing. This is very similar to primary keys in a database schema. OWL is limited in not having any way to define identity criteria that span multiple properties. You can't, for example, define the combination of a person's first and last name as being a joint key.

The lack of unique name assumption is an inevitable consequence of supporting web-scale data integration however it does complicate and limit the use of RDF/OWL. In particular, it affects the ability to validate RDF data. For example, suppose we declare a cardinality constraint that an ont:PhysicalObject can only have a single ont:location value:

```
Ont:PhysicalObject rdfs:subClassOf
   [rdf:type Restriction; owl:onProperty ont:location; owl:cardinality 1^^xsd:int].
```
[13]

Then suppose we are presented with the following RDF statements:

```
urn:server15   rdf:type       ont:PhysicalObject .
urn:server15   ont:location   hp:dataCentre1 .
urn:server15   ont:location   bt:dataCentre2 .
```

Then we can't reject the data as invalid because we can't assume that hp:dataCentre1 and bt:dataCentre2 are different. They might in fact be one place with different URI labels that, for example, HP (who runs the data centre) and BT (who provides the telcoms) know it by. In fact the OWL semantics go further than that and from the data and the cardinality constraint on ont:location we are allowed to conclude that hp:dataCentre1 and bt:dataCentre2 must actually be the same thing. In practice, applications are free to make local unique name assumptions, which are formally equivalent to adding a set of additional owl:differentFrom assertions.

This situation doesn't arise with literal values since we know that "1" is not the same as "2" without further processing.

4.3 The open world assumption

The next major assumption, which is again motivated by the needs of web scale data integration, is known as the *open world assumption* (OWA). The requirement here is that an RDF/OWL processor should not assume that it has all of the data on a given resource - there might be additional statements about it "out there" somewhere else on the semantic web. This means that an RDF store has to be capable of storing additional statements about a resource including ones that use properties which it didn't previously know about and might not be in any ontology accessible to it. It means that an OWL inference engine is not allowed to assume a particular statement is false just because it can't yet find any triples that state (or imply) it.

---

[12] At first glance this sounds wrong because *blank nodes* seem to provide for resources which have no URI. However, strictly a *blank node* actually represents a variable. It is a way of saying "something with these properties exists". The possible values of that variable will be resources with URIs we just don't necessarily know which ones they are.

[13] Here the "[…]" indicate a blank node with the given properties, each property/value pair being separated by a ";".

As we'll see later this assumption has significant benefits for open evolving systems. It means that future extensions to the data models, or integration of unforeseen data sources, are easier because none of the existing processors should be assuming their data sets are complete. However, again there is a cost in terms of validation. Returning to our running example, if we have a restriction that all ont:Server resources must have at least one ont:networkInterface and we are presented with just the statements:

```
urn:server15   rdf:type          ont:Server .
urn:server15   ont:operatingSytsem   "Linux"
```

then we can't immediately claim this data is invalid, despite the "missing" ont:networkInterface property because there might be a statement about that property somewhere else that we haven't yet integrated in. For example, the data we've seen so far might have come from a data source which predates the decision to make the network interface data element mandatory.

4.4 Ontology, schema and objects

One final aspect of the modelling approach is worth discussing in a little more detail – how does an *ontology* differ from a schema or an object oriented design?

At first glance many of the statements that can be made using OWL are quite similar to the sorts of constraints that can be expressed in a syntactic schema language such as XML Schema or in object modelling languages such as UML. They share common features such as type hierarchies and the ability to specify the cardinality of relations and the type values of attributes. However, the critical distinctions become clearer when we consider what the modeller is trying to capture.

When designing an ontology the goal is to capture and model the things that are inherent in the domain with the primary aim of enabling interoperability between different systems. Those things that are irrelevant to the domain itself, such as the serialisation order for a set of attributes, should not be captured in an ontology unless they are true domain invariants needed for interoperability. Conversely constraints and relationships which are important in the domain should be captured in as much detail as possible. The primary actions with an ontology are inference (using the ontology to classify data or infer additional relations) and mapping (aligning different data source schemas with the shared ontology to support data merging).

When designing a schema the goal is to specify how the information being handled should be structured for storage, transmission and programmatic access. The primary aim is to support efficient and safe processing. The schema design removes ambiguity in how the data should be laid out. Constraints that help with efficient access should be captured in detail, constraints that don't help with storage or access are omitted since constraint checking is expensive. The primary actions undertaken with a schema are data validation and access optimization.

These two model types overlap. Some of the invariants of the data structures captured in a schema or object design will be based on fundamental invariants of the domain that would also be captured in a domain ontology. Conversely an ontology is still just a model of the world, it is not some absolute abstract truth, so there are always representation choices to make which are not that different from the representation choices one makes when designing a data or object structure.

Thus it is natural that the languages designed for these different model types do have substantial similarities. However, it can be hard to take a design intended for one use and re-purpose it for the other, because it is hard to reverse engineer the intent behind given modelling decisions. For example, when an XML Schema specifies that the properties of a resource should be ordered in a certain way it is hard to tell if that is an important feature that all data sources will respect (e.g. the order might represent a priority which affects processing outcomes) or is simply there for convenience in defining the data structure.

This comparison also helps us to explain apparent omissions from the language features sets. For example, OWL just has object relationships; there is no distinction between associations and containments as in UML[14]. For data structure design this would be a limitation because it affects storage allocation but this is not such a significant issue for specifying the nature of the domain relationships. Conversely schema languages typically only specify necessary but not sufficient conditions for type membership – they can't be used to enable automatic classification of instance data because that's not relevant for normal schema usage and yet is central to several uses of ontologies.

5 Strengths and benefits

Now that we've clarified the modelling approach itself and the key underlying assumptions we can now look at the specific benefits of using RDF/OWL. In this section we identify and describe those benefits in abstract then in the next section we will illustrate them in more detail with some worked examples.

5.1 Information integration and shared vocabularies

A primary strength of the RDF/OWL stack is the ability to integrate information from multiple, heterogeneous, sources.

The RDF data model itself is inherently composable. Since an RDF data set comprises a set of separate logical statements then combining two RDF datasets just amounts to forming the union of the two statement sets.

The use of global URIs to identify the concepts, relationships and resources means that accidental clashes of names are unlikely and where sources use compatible vocabularies and resource identification mechanisms then useful data merging takes place automatically. Even where different URIs have been used for the same resource then the lack of Unique Name Assumption together with the OWL mechanisms for equality inference (owl:InverseFunctionalProperty, owl:FunctionalProperty, owl:sameAs etc) make it possible to merge the overlapping views of common resources.

OWL provides a standard means to publish a vocabulary (more strongly, an ontology) to allow it to be reused. It is possible for data to reference individual concepts and relations in multiple vocabularies in a fine-grained way. In many applications the development and publication of the shared vocabulary is an important end in itself and can significantly simplify interoperability issues even without the use of the RDF data representation layer.

In cases where the data sources use different vocabularies but the underlying concepts and relations do have some hidden overlap then again the OWL mechanisms for class and property mapping (owl:equivalentClass, rdfs:subClassOf, owl:equivalentProperty, owl:subPropertyOf) allow mappings between the vocabularies to be defined and published.

In the general case, where heterogeneous, independently developed data sources are to be integrated then OWL provides an appropriate language for modelling a shared ontology into which each of the separate sources can be mapped. Except in simple cases, OWL does not itself automate the process of discovering the mappings from one ontology into another - it just provides some tools for expressing and implementing the results of the mapping. However, there are several semi-automated ontology alignment technologies in use or development by various research groups that can assist with mapping development. Many of these tools are targeted at OWL as the open standard appropriate for such work [19].

[14] Interestingly the same is true in the OMG Meta Object Facility, there is a single notion of *association* and other UML concepts such containment and aggregation are then constructed on top.

5.2 Support for semi-structured data

Semi-structured data [7] is data with significant irregularities such as missing attributes, variable numbers of occurrences of attributes and mixed typing. Such data arises naturally when heterogeneous sources are combined and in applications such as knowledge management where there is no guarantee of completeness of metadata mark-up. In semi-structured data problems there is often no *a priori* schema and we have to be able to manipulate data schema-less while later adding (partial) schema information[15] to improve performance and guide later processing.

RDF/OWL is specifically designed to handle such semi-structured data. Each attribute value is a separate RDF statement so any RDF processor can accept, store and query such semi-structured data even in the absence of a schema or ontology. Raw XML is also a semi-structured data representation, though it has a built in bias towards nested tree-structured data and without schema information cannot easily represent arbitrary graph-structured relations in the way that schema-less RDF can.

When we start to provide schema information to express the domain semantics and guide the processing then the differences become clearer. With XML Schema we have something of an all-or-nothing situation. If some attributes may be missing or duplicated due to data source variability then in a XML Schema style we can only express this by leaving such attributes optional. With highly irregular data then everything becomes optional and the schema becomes useless. In contrast, with OWL we can express domain expectations such as a given attribute having a cardinality constraint while the open world assumption means that the RDF/OWL processor will not reject information in which such attributes are missing or in which additional unexpected attributes are present. Despite this default openness the semantic constraints are still expressed and verifiable, so an unexpected additional property which led to a contradiction with existing semantic constraints can still be detected and rejected. Further, RDF/OWL allows us to extract class membership or typing information which is missing from the original data but inferable given the ontology definitions.

5.3 Separation of syntax from data modelling

RDF/OWL allow a data modeller to concentrate on the semantics of the data model (the properties and classes to be used and how they interrelate) and frees them from syntax considerations. Once an appropriate set of vocabulary terms has been defined (typically published as an RDFS or OWL ontology) then a generic RDF processor can immediately serialise and exchange the data, store it in a database, query it and access it though a generic high level API.

In contrast, when working at the XML level one is directly designing a serialization syntax and is forced to make choices (such as use of attributes versus elements and ordering choices) which may not be relevant to the data model itself.

The serialisation syntax itself meets the two primary requirements of an external data representation format [23] – it is self-describing (for example the difference between an integer and a string is clear without the need for external schema information) and supports round tripping.

The separation between the data model and syntax also means that other syntax forms for RDF are available in addition to the standardised RDF/XML syntax. In particular the N3 format [22] is a succinct and convenient language for human authorship and reading of RDF data.

Finally, when processing the data, then operations such as query and transformations are expressed in terms of the data model level not in terms of the document syntax. This can lead to queries and transforms that are more succinct and maintainable than a syntactic equivalent. For example, in a comparison of queries for media repository access [24] RDF

[15] Sometimes called a *data guide.*

queries expressed in SPARQL were found to be less than half the length and more simply structured than corresponding XQuery expressions.

5.4 Web embedding

The aim of the semantic web is to be an extension of the current web. A key feature of the RDF/OWL design that supports this is the use of URIs for all symbols. This has several important benefits.

First, it means that vocabularies can be published on the web, typically at the URL corresponding to the namespace URI for the vocabulary. This encourages reuse of vocabularies including dynamic discovery and loading of reused vocabularies.

Secondly, it means that vocabularies can refer to concepts in other vocabularies at a very fine-grained level. Rather than import an entire ontology one can refer to an individual concept in a published ontology without being forced to commit to the entire ontology which might contain other concepts not relevant to current purpose.

Thirdly, it means that when describing resources that are in fact web resources, such as web services or web publications, there is a natural URI to use. This increases the chances of successfully merging and comparing independently developed descriptions of such resources.

5.5 Extensibility and resilience to change

This same ability to handle irregular and optional data without losing all typing and structure information is also relevant to handling change over time.

A common requirement in many system designs is to allow loose coupling between clients and providers so that the providers can evolve over time without breaking existing clients (backward compatibility) and older providers can successfully respond to updated clients (forward compatibility).

Achieving this resilience to change is simpler using RDF/OWL than using a strict schema-validation approach, particularly due to the open world assumption. For example, if a new style server expects some property P on class C and receives an old style instance of C then the absence of P does not invalidate the data. The server can conclude that a P value exists somewhere (in RDF terms it would be a blank node at this point) and the processor can decide to proceed as is, or investigate an application specific default value. Conversely if an old style server receives a new style model the presence of a previously unknown additional property also does not break the data. It is a valid instance of C as far as an OWL processor is concerned and an RDF processor can continue to manipulate the data even if it doesn't have access to the definition of the new property P. The processor may be able to locate the updated ontology at its URI and so access the definition of P. This would enable it to, for example, look for any application-specific *must understand* flags to decide whether to proceed. As in the semi-structured data case the data can still be processed to some extent even if the updated schema is not available. Furthermore the extension ontology information is still expressed in RDF so it's just data and easy to dynamically load and process - there is no requirement for a compilation-based tool chain to process the schema/ontology versions before they can be effectively used.

Of course it may not be possible to restrict all version changes to maintain direct forward and backward compatibility in which case a transformation process may be required. If many versions may coexist then scaling is improved by mapping each version into an intermediate spanning ontology, avoid the need for $O(N^2)$ pair-wise transforms. The intermediate ontology would only represent the constraints that are invariant in the domain and so should remain fairly stable over time whereas the N version-specific translations would handle additional constraints which that particular version chose to impose.

5.6 Inference and classification, based on a formal semantics

Each of the RDF/OWL component languages has a well-defined formal (model theoretic) semantics, and, despite some technical challenges, the semantics of the layers are compatible and integrated. This formal semantics enables use of RDF/OWL for knowledge representation and enables inference.

Given a set of RDF/OWL statements, the formal semantics defines unambiguously what further statements are entailed or implied by the starting set. For example, we can declare that some property such as sdc:dependsOn is transitive and then an OWL processor will be able to deduce from the statements that A dependsOn B and B dependsOn C that A dependsOn C.

More significantly when we define an OWL class we can make that definition complete in the sense that any resource which meets the description of the class can be deduced to be a member of that class. For example, we can define a Father as any member of the class Person who is also a member of the class Male and has at least one value for their hasChild property. This enables RDF/OWL to perform classification tasks over symbolic data and this capability is used in areas ranging from medical diagnosis [8] to service discovery [9].

OWL is based on the branch of logic known as Description Logic, which is aimed at classification and taxonomic processing. It is a very expressive description logic (specifically SHOIN[10]).

5.7 Representation Richness

The RDF data model is essentially a labelled graph with no restriction on cycles. This makes it well suited to representation of complex relational information such as service dependency graphs or server topology diagrams.

XML natively represents hierarchical, tree structured data, but is clearly also able to encode graph information given an appropriate data model. RDF simply has this ability "built in" so that no schema is needed to identify the graphical structure being encoded and some of the OWL inference capabilities (such as the transitive relation example mentioned earlier) can be useful in processing such graph structured data.

This representation richness carries over to the associated query language SPARQL [25], which is well suited to the querying of graph structures.

5.8 Provenance

RDF/OWL has built in tools for annotating RDF statements in order to encode things like provenance information. This is useful in applications where data is drawn from several sources but any conclusions from the data need to be traceable back to the underlying sources.

For example, the SWED [21] application is a distributed directory of environmental organisations. Each organisation self-publishes a machine-readable description in RDF and other parties can add additional statements about organisations (including inter-organisation relationships and classification of organisations according to specialist taxonomies). The portal aggregates this distributed set of information sources and provides a browsable view onto the space. When displaying the information on a given organism we integrate the organisation's own description and the externally supplied information into a single page but provide user interface clues and trace-back facilities to enable the source (and hence trustworthiness) of each statement to be determined.

Again other formalisms can be used to encode such information it is just that RDF includes a built in mechanism, reification, to support it; though the vocabulary which is used for the annotations themselves is open.

5.9 Combining instance and model information

The current languages such as RDFS and OWL that layer on top of RDF are themselves expressed in RDF. This means it is very straightforward to embed modelling information along with instance data. Thus when a server reports data which uses extensions to the common vocabulary it can send the OWL definitions for those extensions along with data – helping the client to decide whether and how to process the extensions.

In RDFS and OWL/full this meta-level mixing goes further in that a single resource can be both an instance level resource and a model-level resource (e.g. a Class). This lack of separation between meta-levels opens up a wider range of modelling styles. For example, consider the question of modelling server operating systems. A natural modelling style is to define a property hasOperatingSystem that relates a server instance to the type of operating system it is running. In that case an individual type of operating system such as Linux is being treated as an instance level symbol. However, we will often also want to record that Linux is actually a family, Class, of operating systems that is a sub-class of Unix and super-class of Fedora, FedoraCore3 and so on. The ability to view the same symbol from multiple modelling perspectives can improve the clarity of models in such cases while not precluding modelling styles which maintain a strict separation of meta-levels.

6 Simplified illustrative example

To make the above discussion less abstract let's take a look at a small example of a modelling problem and illustrate how some of the features we've identified work in practice.

As an example problem we will look at one drawn from the domain of Service Delivery Controllers (SDC) [16]. The details of the SDC architecture are not relevant to understanding the example. The main background to be aware if is that SDCs form an ecosystem in which different systems (indeed different vendors) expose and manipulate abstract models of managed resources. The abstract models provide loose coupling between systems so that a client does not need to be exposed to internal details of a given SDC. It is important that SDC providers retain the freedom to innovate. Due to the distributed nature of the architecture this means that SDCs should be able to upgrade their capabilities (and thus the models they expose) without invalidating existing clients. At any time within the SDC ecosystem different versions of a given SDC category may coexist. The modelling approach thus needs to be resilient to such changes.

To keep the modelling problem small enough to cover in this note we'll just look at the very simple case of an SDC for managing a (virtualized) storage volume. An XML Schema design for the core of such an interface model is[16]:

```
<xs:complexType name="StorageTemplateType">
   <xs:sequence>
      <xs:element ref="storage-t-xs:Size" />
      <xs:element ref="storage-t-xs:RaidLevel"  />
      <xs:element ref="storage-t-xs:Hosts" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:element name="Size" type="xs:unsignedInt" />
<xs:element name="RaidLevel" type="xs:unsignedInt" />
<xs:element name="Hosts" type="storage-t-xs:HostType"/>

<xs:complexType name="HostType">
   <xs:sequence>
      <xs:element name="WWN" type="xs:string" minOccurs="1" maxOccurs="unbounded" />
```

[16] This example has been drawn from an example in version 1.0 of the SDC Toolkit but omits the standard boiler-plate MUWS properties for brevity.

```
    <xs:element name="OSType" type="xs:string"/>
   <xs:element name="Name" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
```

How would we model this differently in an RDF/OWL style? On such a simple example the differences will be small but we can use those small differences to illustrate some of the principles.

First, ideally we would expect to be able to reuse existing ontologies to represent common concepts such as HostType or OSType and the SDC model would only need to draw those together. Since those don't exist at the moment we have to model those concepts ourselves but we'll split them into groups to make it simpler to reuse them elsewhere.

Secondly, we would use typed symbolic values in place of strings to represent values such as RaidLevel. That allows us to extend the range of allowed symbols in the future without having to alter each schema that uses it. It also allows us to attach properties to the symbols for documentation or user interface purposes (for example a reference to a web page describing the nature of the raid level) or to assist future machine processing (for example the minimum number of drives required or application classes for which the raid level is recommended). This symbolic representation style is perfectly possible in XML directly, but RDF/OWL encourages and facilities it.

Similarly, it would be preferable to express numeric values such as size using an explicit shared ontology of units. It would not be worth developing a units ontology just for this SDC but an adequate units ontology of basic computing terms would be likely to be highly reusable.

Thirdly, we would explicitly model ways of identifying the resources, to assist with data merging. For example, we might allow a Host to have a network address and regard the IP address as an identifying property, which allows us to link to other views of the same host. For this we might be able to reuse an ontology designed to give an appropriate view such as the service discovery ontology [17].

Fourthly, we would not include data layout constraints that are not fundamental to the domain. In this simple example the ordering (xs:sequence) of the properties on StorageVolumeTemplate and HostType are arbitrary and wouldn't need to be modelled.

In general we would look for places where inference such as instance classification or transitive closure might be used in the application and design the ontology to support it. In this simple example there is limited use for this.

That leads us to a model design with two supporting ontology (one GeneralComputingConcepts ontology and one ComputingUnits ontology) together with the StorageVolume ontology.

In outline the GeneralComputingConcepts ontology we need would look something like this[17]:

[17] We're using an information graphical notation to convey the basic idea rather than give a formal machine readable design. We're using ovals for resources and boxes for classes. Property arrows on classes indicate a property with a domain or range indicating that class whereas a property arrow on a resource represents an instance level relationship.

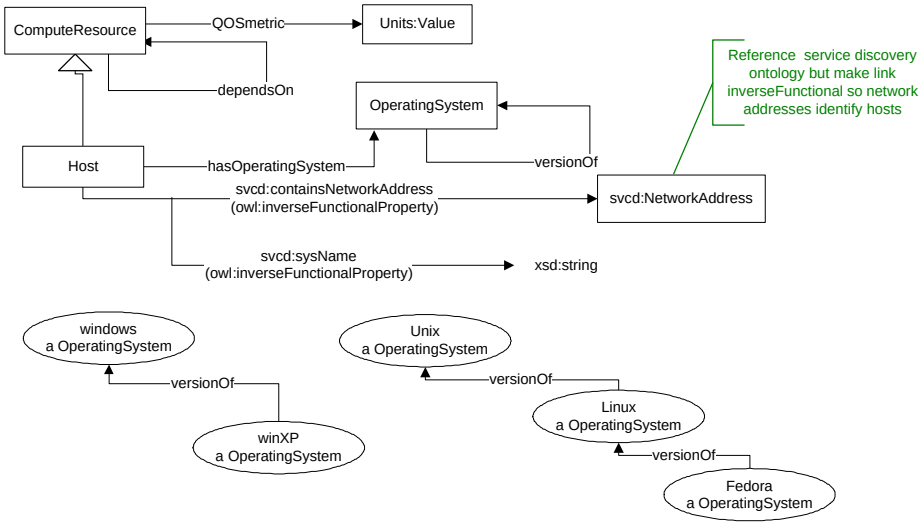


Figure 2: Illustrative GeneralComputingConcepts ontology

A real reusable GeneralComputingConcepts ontology would be rather more comprehensive.

Note that we are able to declare generic concepts and specific reusable instances (e.g. general:Linux as an instance of general:OperatingSystem) in the same ontology document.

Similarly the fragment of the ComputingUnits ontology might look like:

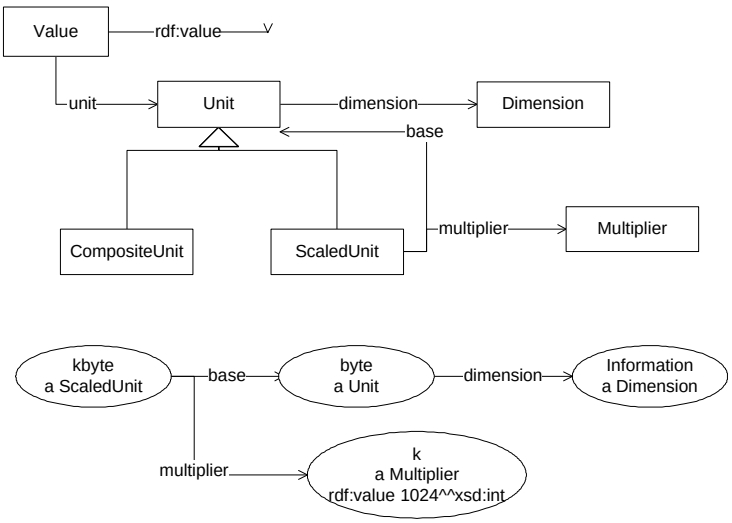


Figure 3: Illustrative Units ontology

The StorageVolume ontology itself might look like:

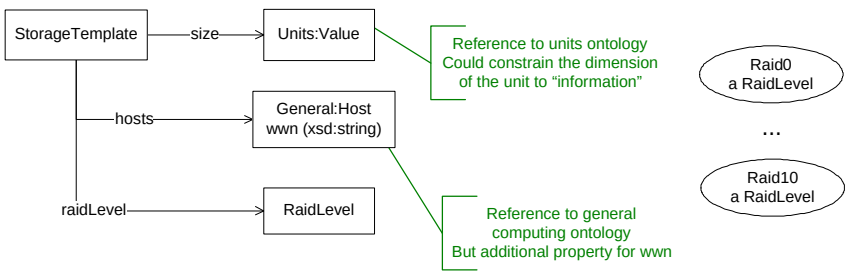


Figure 4: Illustrative StorageTemplate ontology

An instance of such a StorageTemplate in RDF/XML would look like:

```
<rdf:Description rdf:about="#exampleStore">
  <rdf:type rdf:resource="&store;#StorageTemplate" />
  <store:size>
    <units:Value>
      <rdf:value rdf:datatype="&xsd;#integer">42</rdf:value>
      <units:unit rdf:resource="&units;#kb"/>
    </units:Value>
  </store:size>
  <store:raidLevel rdf:resource="&store;#raid0"/>
  <store:hosts>
    <general:Host>
      <rdfs:label>Host 1</rdfs:label>
      <svcd:containsNetworkAddress rdf:resource="#netAddress3"/>
      <store:wwn>WWN1</store:wwn>
    </general:Host>
  </store:hosts>
</rdf:Description>
```

The differences in model between the handcrafted XML and the RDF instance models look quite modest, so what are the consequences of the approach for interoperability between SDCs or resilience to change? We'll look at a series of simple example changes or integration scenarios to illustrate the effects.

First, consider the situation where a provider wishes to offer an SDC with slightly enhanced capabilities, for example it might support a new combination raid level (e.g. 0+1). It can advertise the new raid level by including the instance raid0plus1 in its query template with a declaration that it is an instance of the RaidLevel class. Any template instance that uses this remains a valid instance of the existing StorageVolumeTemplate class. This exploits the open extension of classes together with the ability to mix ontology additions in with instance data. It would also be possible for the extended SDC to publish the ontology extensions separately at a new URL and use the rdfs:isDefinedBy property to embed a reference to this extension in the instance documents. The existing ontology documents do not need to be changed.

Note also that forward, as well as backward, compatibility is supported in this case. If an updated client requests raid0plus1 from a non-updated SDC then the SDC still receives a syntactically valid request and the type inference will determine that the newly requested raid0plus1 must be a RaidLevel. That allows the server to at least respond with a "can't meet request" message, perhaps including the list of supported raid levels, rather than an "ill-formed request" message.

Similarly the updated SDC could offer a more precise reporting of the OS of the storage host (e.g. FedoraCore3) and use the general:versionOf property in the upper ontology to enable clients to know that this is just a variant on Linux.

A related resilience is achieved by the use of explicit units. There is no ambiguity over whether size is being measure in bytes, Kbytes or whatever; nor whether we are using k as 1000 or 1024. Assuming the units ontology were supported by a simple ontology-driven converter tool then clients would be free to express storage in the appropriate units and SDC instances would convert to native units.

What if the enhanced SDC wanted to offer a completely new configuration or reporting parameter, for example a response-time measure? Requesting the observed state model of the enhanced storage SDC would include a new property (say store:responseTime) alongside the existing properties. This would remain a legal StorageTemplate so any client would still accept it as a valid instance. The additional properties would flow through any RDF-based exchange or storage mechanisms so that if another client in the eco-system

were able to understand the new property it would remain available despite the existence of non-upgraded intermediaries.

In a syntactic schema approach this extensibility would require all schemas to include an open-ended options area (e.g. an xs:any element). However, in the RDF/OWL case the built-in open extension, which derives from the open world assumption, still permits structure and typing information to be accessed and used by non-upgraded clients. For example, the SDC could declare the new store:responseTime property to be a subPropertyOf the general:QOSMetric property in the GeneralComputingConcepts ontology. In that way unmodified UI clients could display the QOS parameter appropriately and SLA monitors could set and check the property value.

These examples illustrate how reasonable extensions to the model can be made incrementally retaining a useful measure of forward and backward compatibility. There is no magic here. If the interface model changed radically (for example using different property names or changing the structure) then existing clients would need upgrading. In cases where the existing functionality is still supported then an intermediary could be used to transform between the old and new model structures [18][11].

Finally, consider the situation where one client is able to view data from multiple overlapping sources. For example, the Service Discovery (SVCD) tool is able report an RDF description of discovered nodes and services such as:

```
<svcd:Application rdf:about="http://localhost/data#oracle1">
  <oracle:hasInstance>
    <oracle:Instance>
      <svcd:name>Payroll</svcd:name>
    </oracle:Instance>
  </oracle:hasInstance>
  <svcd:appType>Oracle</svcd:appType>
  <svcd:hostedOn>
    <svcd:Node>
      <svcd:containsNetworkAddress rdf:resource="#netAddress3"/>
    </svcd:Node>
  </svcd:hostedOn>
  <svcd:observedOn rdf:resource="#netAddress3"/>
</svcd:Application>
```

A generic RDF/OWL processor can merge this data with our StorageVolume observed state based on the identity criterion for network addresses[18]. This tells us that the Host for this storage volume is also the Node discovered by SVCD and so hosts the Oracle application that contains an Oracle instance called "Payroll". This merging can continue across other data sources so long as there are some shared ontology components such as identifying properties (or common URI allocation algorithms), for example [11] shows merging between SVCD and SOAManager data.

To keep this section short and accessible we've deliberately chosen a simple example. We make no claims that this sort of example can't be handled in other ways, nor that the same modelling idioms couldn't be adapted to other modelling languages. As explained earlier, an ontology differs from a schema due to what it is trying to represent rather than due the language it is expressed in. However, hopefully the example is concrete enough to illustrate something of how RDF/OWL works and the modelling style it encourages and facilitates.

[18] For brevity we are simplifying here. In practice the NetworkAddress object is a blank Node which in turn can be identified though uniqueness of the associated IP address.

7 Weaknesses, costs and trade-offs

Any language design involves trade-offs between different competing requirements. We've seen that RDF/OWL particularly supports the requirements of distributed information processing – integration, evolution, handling semi-structured data. Now we look at that cost of that support, the areas in which RDF/OWL is weaker as a consequence.

7.1 Validation

The most obvious weakness of RDF/OWL is the limited data validation that is supported. OWL can express typical constraints such as cardinalities and ranges that are used in schema-based data validation. However, the open world assumption and no-unique name assumption limit the value of these declarations for data validation. The open world assumption means that having too few properties on a resource does not necessarily violate a minimum-cardinality constraint. Similarly, the no-unique-name assumption means that having too many object properties on a resource does not necessarily violate a maximum-cardinality constraint unless those resources are known to be distinct.

This weakness is the direct trade-off made to achieve the resilience to change and data integration support described earlier. There is, in fact, nothing to stop an RDF/OWL processor making a local closed world assumption and/or unique name assumption and validating a given document against the class declarations as if it was a closed and complete specification[19]. However, doing so then sacrifices some of the extensibility benefits. For instance, in our example in section 6 if we had validated the raid level values against the closed set of known raid level types then we would not be able to accept a new locally introduced vendor-specific raid level type without making global changes.[20]

Whilst the validation support is limited it is not zero. Typing information can still be checked when types are known to be disjoint, maximum-cardinalities can be checked for values which are known to be distinct (such as literal values), simple data type limits can still be expressed and checked (using XML schema datatype definitions).

Over time we hope that tools and best practices to support expression and testing of additional validation criteria will become more widespread. We can then see that the separation between the model and the validation criteria is useful. It enables us to have several systems share the same underlying model but apply different validation requirements. For example, one system may be flexible and able to adapt to missing elements, another may require a complete closed model in order to proceed.

7.2 Expressivity limitations

OWL walks a fine line between expressivity and tractability. The consequence of a using a logic-based knowledge representation approach is that inference is a more expensive operation than schema-compliance checking. Within the space of description logic languages OWL/DL is as expressive as it can be without being undecideable. Several expressivity limitations affect the modelling power of the language in significant ways.

Cross-slot constraints and operations – OWL is only able to express restrictions on a single property at a time. You can state that on class C property p has a certain value, certain number of values or certain range of values. You can't express that the value of property p1 is greater than the value of property p2 or that the value of property p3 is the sum of the values of p1 and p2.

---

[19] Indeed HP offers an extensible validation tool, called Eyeball (http://jena.sourceforge.net/contrib/contributions.html) which supports plugin closed-world validators.

[20] There are compromise design patterns open here. One could have a centralised ontology of raid levels published at some agreed URI, separate from the ontology defining the StorageTemplate. Raid levels could be validated against this controlled ontology and accepted new raid levels could then be dynamically published via updates to the centralised ontology.

Identity criteria - A related consequence is that resource identity criteria are also limited to one slot at a time. It is not possible to express the equivalent of composite keys in ER modelling.

Property composition – OWL cannot express the fact that one property is the composition of two other properties. For example, it is not possible to define an uncle relation as the composition of brother and parent.

Defaults – OWL can't be used to describe default values for a property. This is another corollary of the open world assumption. If a processor assumed, because it hadn't yet seen a value for a property, that it should use a default instead then if at a later time the correct value arrives there would be a problem. OWL is declarative and monotonic; an OWL processor should only draw conclusions that would remain valid if additional statements were added to the data set.

These limitations could mostly[21] be overcome by the use of a rules language to complement OWL for expressing relations between RDF resources. Such a language may well be the focus for the next round of standardization [12].

7.3 Performance

The open extensibility of RDF has performance costs. Just as the relational database model is more expensive to query than network or hierarchical models so the general RDF triple model is more expensive to query than structured objects would be. If the data being processed is semi-structured anyway, or will become so as sources and structures evolve, then there may be little loss. If an application has well-structured data with stable, unchanging schemas then higher performance could be achieved with a less general data representation. The success of the relational database model over the older network and hierarchical models suggests that extensibility is valuable in general, though non-relational databases continue to exist in niche areas such as telecoms billing where performance is critical. However, the implementation technology for efficient triple stores is still maturing and the precise performance overheads for large scale stores are not yet clear.

A related issue is that of storage cost. The flexibility of the RDF data model means that high performance stores require significant indexing and the size of the indexes can exceed the size of the data. The data itself is, in any case, inherently bulky due to extensive use of URIs (though namespace compression helps) and the need for inline typing information for literals. This may be an issue for processing of RDF on memory limited devices.

At the ontology layer, clearly inference is a more substantial process than schema checking and performance of OWL inference can be a barrier in some applications. As noted earlier, OWL/DL is decidable but does have a very high complexity class (NEXPTIME). For many subsets of it algorithms are known which are tractable in practice.

7.4 Serialization issues

Whilst the serialization format for RDF/OWL is XML the flexibility required by RDF causes some problems with the XML representation. The XML syntax for RDF is defined by a W3C specification and implemented in a number of RDF parsing tools but cannot be expressed within the limitations of the XML Schema language. This can make it hard to use RDF data with systems that mandate an XML Schema specification.

The conflict between the open world requirements and tight schema verification means that some mismatch is inevitable but RDF/XML is worse than it need be. The

[21] The first three can be handled this way. Rule languages can indeed handle the last one, defaults, but not without violating the open world assumption. It is an open question whether and how a semantic web rule language would handle scoped closed world assumptions.

serialization format makes use of QNames to represent properties which makes it impossible to separate the general syntactic schema from the specific RDF vocabulary. Further, the RDF/XML specification includes a number of "abbreviation rules" intended to make it simpler to treat a typically XML syntax as if it were legal RDF/XML. Unfortunately, this means that a given RDF data model may be serialized in several ways that are semantically equivalent but syntactically different. Processing such data with a syntactic tool such as XSLT is difficult. This also makes it hard to use XML development aids such as validating editors or form designers with RDF/XML data.

There are various proposals for simplified XML representations [15] which might overcome such difficulties at the expense of succinctness and readability of the serialization.

7.5 Familiarity

Finally, one of the biggest weaknesses of the RDF/OWL stack at present is its relative unfamiliarity. The terminology itself (e.g. "ontology") is off-putting. The background theory is inaccessible to most developers (few will, for example, have come across the notion of Description Logic) and the existing literature makes it unclear how much of this background theory is necessary in order to develop successful applications. The modelling style (symbolic logical models) is sufficiently different from that which most software developers encounter to impose a substantial learning curve.

The fact that parts of the modelling (the notion of classes and properties) appear somewhat similar to the more familiar notions of object oriented designs is a mixed blessing. It can encourage developers to see the languages as mere variants on an object-oriented approach. This can reduce the apparent initial learning curve but eventually the detailed differences in semantics will be apparent. Furthermore it may lead to designs which miss the benefits of resilience and data integration by encoding arbitrary data layout and processing assumptions into the domain models.

This barrier could be reduced through the development of appropriate training resources.

7.6 Continuous domains and uncertain or contradictory data

RDF/OWL is a logical symbolic representation. It has no provision for representing uncertain or continuous data.

It is possible to design representations for uncertainty and embed them in RDF/OWL. For example, one could create a vocabulary to describe a belief propagation network such as a Bayesian network and then encode and transmit such Bayesian networks as RDF data. This would enable different systems based on such networks to exchange their beliefs using a RDF-based exchange data model. However, the machinery of RDF/OWL would not itself represent the semantics of the belief propagation algorithms and there would be no useful relation between the concepts encoded in the belief network and concepts in an OWL domain ontology.

Similarly the categories represented by OWL classes are discrete things. A resource is either a member of a given class or it is not. In data management domains where modelling technologies such as ER modelling are applicable then this symbolic approach is entirely appropriate. However, in other domains such as legal or political decision support the categories are often continuous and overlapping and not suited to OWL modelling.

The discrete, logical nature of OWL modelling has a further consequence, the inability to handle contradictory information. If a set of OWL statements implies both *P* and *not P*

(for some statement *P*) then there is a global inconsistency and no useful deductions can be made at all.[22]

7.7 Representation of processes and change

RDF/OWL are declarative languages for expressing an information model they are not procedural languages for expressing algorithms or processing models. So for, example, they do not have any built-in notions of time, state or process and no built in semantics for reasoning over dynamic process models.

Again it is possible to create ontologies for statically describing such notions. For example the Semantic Web Services Framework (SWSF) [13] provides an ontology for describing processes based on PSL [14]. However, it was necessary to introduce an expressive rule language alongside OWL in order to express the axiomatic constraints of PSL. Certainly concepts such as deadlock detection are well outside the scope of generic RDF/OWL processors.

8 Combining RDF/OWL with XML and other formalisms

In this paper we have looked at the nature of RDF/OWL models and the strengths and weaknesses of the modelling approach. In doing so we have made several comparisons to alternative formalisms such as XML Schema. In this final section we will briefly consider the ways in which RDF/OWL can be combined with other formalisms. A comprehensive examination of this issue would require a paper in its own right so we will restrict ourselves to looking just at combinations with XML Schema.

XML Schema offers strong validation, efficient processing and is in wide spread use. Hence it is useful to consider how XML Schema data could be combined with RDF/OWL models.

RDF provides two basic mechanisms that can be used to relate RDF models to non-RDF XML – URIs and XML literals. First, if the XML data can be identified by a URI (for example via a document URL or an XPointer URI reference) then it can be treated as any other web resource and be the subject and object of an RDF expression.

Secondly, RDF provides a mechanism for embedding well-formed XML fragments within an RDF model. At the syntactic level this facility is supported in RDF/XML by a rdf:parsetype="Literal" declaration which causes the XML sub-tree to be subjected to XML Exclusive Canonicalization and converted to a typed literal of type rdf:XMLLiteral. This allows an XML fragment to be used as the object of a RDF statement (i.e. the value of some property). RDF does not permit literals in the subject position of triples so if one needs to make RDF statements about an XML fragment then (apart from the URI Reference approach) one has to do so indirectly. For example, by defining a blank node to represent the XML fragment and making statements about the blank node.

These mechanisms are useful for several tasks. First, they allow us to embed expressions in other XML standards within a RDF model. For example it is possible embed MathML expressions with the RDF description of a resource to express algebraic or logical relationships that are outside the scope of OWL. Second, they allow us to use RDF to provide metadata annotations about an XML structure or relational links between structures.

There are some limitations to this however.

Firstly, this is a largely syntactic device. There is no direct relationship between the RDF/OWL semantics and any semantics associated with the XML. For example, in an OWL model we can declare the range of a property to be rdf:XMLLiteral but cannot

[22] There are logics in which contradictions do not have such a global effect but they are beyond the scope of existing semantic web standards, and of this paper.

declare it to be an instance of a particular XML Schema complex type[23]. Since the RDF datatype system is open it would be possible to define a system of RDF datatypes that did support such declarations but there is no such definition in the existing standards.

Secondly, there is relatively little current tool support for working with mixtures of RDF and non-RDF XML. The schema-unfriendly nature of RDF/XML discussed earlier makes it hard to apply general XML tools such as validating editors to the hybrid RDF/XML documents. Existing RDF processors support ingest, storage and access to the embedded XML fragments but do not typically support non-standard extensions such as ability to validate the embedded fragments against an XML Schema.

9 Summary of applicability

In summary, RDF/OWL is particularly suited to modelling applications which involve distributed information problems such as integration of data from multiple sources, publication of shared vocabularies to enable interoperability and development of resilient networks of systems which can cope with changes to the data models.

It has less to offer in closed world or point-to-point processing problems where the data models are stable and the data is not to be made available to other clients. It is also largely unsuited to domains involving continuous or fuzzy categories.

10 Acknowledgements

Grateful thanks to many people for useful review comments on earlier drafts of this paper especially Mark Butler, Martin Merry, Craig Sayers and Paul Shabajee.

Some of the material derives from our work with Derek Coleman, Nigel Cook and Bryan Murray on the SDM community and from discussions with other RDF users in HP notably Dave Scott Roth, Brooks Bollich, Jon Lachelt, Tilo Nitzsche and David Booth. Thanks to all.

11 References and further reading

[1] F. Manola, E. Miller (editors), RDF Primer, W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-primer/

[2] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993.

[3] Brickley D., Guha R.V. (Editors), RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, 10 February 2004. http://www.w3.org/TR/rdf-schema/.

[4] Sean Bechhofer et al (Editors), OWL Web Ontology Language Reference. W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-ref/

[5] The Gene Ontology. http://www.geneontology.org/

[6] The National Cancer Institute Thesaurus in OWL. http://www.mindswap.org/2003/CancerOntology/

[7] Dan Suciu, An Overview of Semistructured Data. SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory). Volume 29, number 4, pp 28-38. 1998.

[23] RDF does support XML Schema simple types and their restrictions.

[8] Semantic web for Pathology project. http://www.inf.fu-berlin.de/inst/ag-nbi/research/swpatho/english/projectdescription.htm

[9] D. Trastour, C. Bartolini, J. Gonzalez-Castillo, A Semantic Web Approach to Service Description for Matchmaking of Services. Proc. 1st Semantic Web Working Symposium, CA. 2001.

[10] I. Horrocks, P. F. Patel-Schneider. Reducing OWL Entailment to Description Logic Statisfiability. J. of Web Semantics, 1(4):345-357, 2004.

[11] Tilo Nitzsche. RDF Proof of Concept project. RDF Model Exchange between SOA Manager and OVO/ServiceNavigator.

[12] Rule Language Standardization. Report from the W3C Workshop on Rule Languages for Interoperability. http://www.w3.org/2004/12/rules-ws/report/

[13] S. Battle et al, Semantic Web Services Framework, 2005. http://www.daml.org/services/swsf/1.0/overview/

[14] M. Gruninger. *A Guide to the Ontology of the Process Specification Language. Handbook on Ontologies in Information Systems.* R. Studer and S. Staab (eds.). Springer Verlag, 2003.

[15] Carroll, Jeremy J.; Stickler, Patrick. TriX : RDF Triples in XML. HP Laboratories Technical Report. HPL-2004-56

[16] D. Coleman, C. Thompson, Model Based Automation and Management for the Adaptive Enterprise. HP Openview University Association, 12th Workshop. Porto Portugal. July 2005.

[17] J. Lachelt, Service Discovery Data Vocabulary: Functional Specification. Version 0.4. May 26th 2005.

[18] D. Reynolds, et al., Use of semantic web modelling in MAE architecture. Forthcoming.

[19] Y. Kalfoglou, B. Hu, D. Reynolds and N. Shadbolt. Semantic Integration Technologies Survey. CROSI project, 6th month deliverable. University of Southampton, Technical Report, E-Print No #10842. May, 2005. Available from: http://eprints.ecs.soton.ac.uk/10842/

[20] J. Carroll et al., Jena: Implementing the Semantic Web Recommendations, HP Laboratories Technical Report, HPL-2003-146, 2003. http://www.hpl.hp.com/techreports/2003/HPL-2003-146.html Software and documentation available from: http://jena.sourceforge.net/

[21] D. Reynolds et al., SWAD-Europe deliverable 12.1.7: Semantic Portals Demonstrator- Lessons Learnt, http://www.w3.org/2001/sw/Europe/reports/demo_2_report/

[22] Tim Berners-Lee, Notation 3, http://www.w3.org/DesignIssues/Notation3.html

[23] J. Siméon, P. Wadler, The Essence of XML, POPL'03, January 15-17, new Orleans, USA. http://homepages.inf.ed.ac.uk/wadler/papers/xml-essence/xml-essence.pdf

[24] M. Butler, Comparison of SPARQL and XQuery for sample media queries. Personal communication.

[25] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF, W3C Working Draft 21 July 2005, http://www.w3.org/TR/rdf-sparql-query/