Igniting the OWL 1.1 Touch Paper: The OWL API

Matthew Horridge[1], Sean Bechhofer[1], and Olaf Noppens[2]

[1] The University of Manchester
[2] Ulm University

Abstract. This paper describes the design and implementation of an OWL 1.1 API, herein referred to as the OWL API . The API is designed to facilitate the manipulation of OWL 1.1 ontologies at a high level of abstraction for use by editors, reasoners and other tools. The API is based on the OWL 1.1 specification and influenced by the experience of designing and using the WonderWeb API and OWL-based applications. An overview of the basis for the design of the API is discussed along with major API functionality. The API is available from Source Forge: http://sourceforge.net/projects/owlapi

1 Motivation

The broad acceptance of the forthcoming OWL 1.1 ontology language will largely depend on the availability of editors, reasoners and numerous other tools that support the use of OWL 1.1 from a high-level/application perspective. Building such tools usually requires an easy to use API so that developers are divorced from the issues of parsing and serialising particular syntaxes, which allows them to concentrate on manipulation of ontologies at a high level of abstraction.

Looking at the wider picture, a common application infrastructure can pave the way for easy access to and integration of ontology management services. For example from versioning and diffing services, through to reasoning and inference services such as explanation and debugging. This was illustrated in [1] where the development of real-world applications essentially highlighted the need for a common underlying infrastructure.

For the past three years, since the emergence of OWL 1.0, the WonderWeb API [2] has met needs of application and tools developers. It has been used as the foundation for popular editing tools such as Protégé-4 [3], Swoop [4] and OntoTrack [5], and for various patching, reasoning and validation services such as the ever popular WonderWeb validator[3]. With over 2000 downloads per release, the demand for such an API has been proven.

With this in mind, and the experience of developing a raft of the afore mentioned tools, we have created an API for manipulating OWL 1.1 ontologies. The core API provides functionality for creating, examining and modifying OWL 1.1 ontologies. It also ships with a selection of parsers, renderers, and interfaces to

[3] http:://phoebus.cs.man.ac.uk:9999/OWL/Validator

various state of the art reasoners. Moreover, in a bid to improve the experience of application development and to obviate the need for re-implemention of common functionality, the API had been augmented with additional components. Amongst other things, fine-grained access to ABox query results, black-box debugging, publish-subscribe mechanism for TBox changes, and validators and detectors for the official OWL 1.1 fragments[4] have been provided. Although the OWL API has stemmed from the well known WonderWeb API, there have been significant changes to the interfaces. In order to get a feel for compatibility issues, the next section describes the differences between the WonderWeb API and the OWL API .

2 From the WonderWeb API to an OWL 1.1 API

The WonderWeb API was developed as part of the WonderWeb project[5] in order to "help realise the vision of the Semantic Web". The primary goals of the API were to push OWL and provide a highly reusable component for the construction of different applications. The API has several noteworthy features:

- Close correspondence with the OWL 1.0 Abstract Syntax[6] – incorporating both a frame style and an axiom oriented approach (or even a combination of both).
- Syntax neutrality – the API is not biased towards a particular concrete syntax.
- Support for multiple ontologies – all queries and changes are made with respect to an ontology or a set of ontologies.
- First class support for changes – ontology changes are applied through change objects which allows clear and easy change tracking.

The first point in the above list is worth more than mentioning in passing: A feature of the OWL 1.0 Abstract Syntax specification is that it allows class descriptions to be specified in a frame style syntax or in an axiom oriented syntax. For instance, the class definition $ClsA \sqsubseteq ClsB \sqcap ClsC$ can be written as either `Class(ClsA partial ClsB ClsC)` (using a frame style syntax), or as `SubClassOf(ClsA ClsB), SubClassOf(ClsA ClsC)` (using an axiom-oriented syntax). A fundamental design philosphy of the WonderWeb API was that the abstract syntax was closely parallelled by the API interfaces. To this end, the distinction between a frame based and axiom based modelling style is preseved at the API level. Like the Abstract Syntax, the API tends to bias the user towards a frame style of modelling.

In October 2006, motivated by the DIG 2.0 working group[7], a revised OWL 1.1 specification was released. A major change from initial drafts was that the

[4] http://owl1_1.cs.manchester.ac.uk/tractable.html
[5] http://wonderweb.semanticweb.org/
[6] http://www.w3.org/TR/owl-semantics/
[7] http://dig.cs.manchester.ac.uk/

frame style syntax was dropped in favour of a *completely axiom oriented syntax*. In addition, the use of UML presented a clear structural definition of what it meant to be an OWL 1.1 ontology. Since much of the WonderWeb API was based around the OWL 1.0 frame style syntax, it became clear that whole API would have to be migrated to an entirely axiom oriented representation. In other words, it was no longer feasible to make simple extensions to the API in order to provide OWL 1.1 support. In practice this meant taking the best design aspects of the WonderWeb API, refactoring the interfaces and providing a completely new reference implementation. With interface changes and the move towards axioms, porting an existing WonderWeb API application to the new OWL 1.1 API may seem like an unecessary burden. However, it is hoped that this will be ameliorated by the design and structure of the API closely following the OWL 1.1 specification so that the translation from specification to API is a simple one.

3 Design decisions and API features

When developing OWL 1.0 APIs there are a number of ways in which developers can take the specification and produce a set of interfaces for manipulating OWL 1.0 ontologies. The OWL 1.0 documentation does not make any strong suggestions about what an OWL 1.0 implementation should look like. In contrast, the OWL 1.1 documentation is much more clear about what a "standard" OWL 1.1 implementation should look like. In fact, there is very little ambiguity with regards to how ontology constructs should be translated into API interfaces. While this does not close the door to alternative designs, the effect is that implementors are steered towards the same end point.

In essence the specification is well presented, clear and concise and this makes designing an OWL 1.1 API a relatively straight forward and easy process. Because of this, the OWL API interfaces are more or less a carbon copy of the interface descriptions found in the OWL 1.1 specification. The following sections detail some of the more important aspects of the design of the API.

3.1 Axioms everywhere

The OWL 1.1 specification views an OWL ontology as a set of axioms. In the API, an `OWLOntology` interface provides access to the axioms in an ontology. Not only does this include logical axioms, which encompass traditional OWL-DL axioms and also rule axioms, but it also includes annotations which are themselves represented as axioms.

In practice, the benefits of having an axiom based representation cannot be overstated. Direct experience of developing tools such as expressivity checkers, species validators and translators for reasoners suggest implementations are *much* cleaner when dealing with axioms. Consider the task of comparing two different versions of OWL ontologies using an axiom based approach when compared to a frame based approach or even RDF – when using axioms a simple comparison boils down to a set difference operation. As further evidence, experience of the development of debugging tools, which use black box debugging algorithms developed by Kalyanpur [6], suggests that an implementation using anything other than direct manipulation of axioms would have been painfully messy and verbose.

The API provides methods for the filtering and grouping of axioms in an ontology in an attempt to support as many different types of applications as possible. Axioms may be retrieved by entities that the axioms describe/define, entities that reference them, and the type of axiom amongst others. For example, given a class, it is easy to obtain the subclass, equivalent and disjoint class axioms that define the class. It is also easy to obtain the axioms that merely reference the class, such as domain and range axioms or class assertions. Such filtering and indexing makes it particularly easy to obtain the context of usage for a given entity. Additionally, graphical ontology editors such as Protégé and Swoop tend to use frame based presentations. In these situations, the provision of axiom filtering and grouping methods, which can be used to compose frame based views for any entity is vital. For example, consider the presentation of the description of a class in an editing environment. Imagine that the display contains information relating to the selected class, for example annotations, superclasses, subclasses, equvialent classes, disjoint classes, instances and so on. The editing tool must have some method of selecting the required axioms from the entire set of axioms in the ontology in order to compose such a view. Not only does the provision of a suite of methods that allow such frame based views to be composed reduce the implementation burnden for tools developers, it also gives them hope that such views can be composed in an *efficient* manner.

3.2 Concrete Representations

OWL has a variety of concrete representations, the most widely used being RDF. In fact one could be forgiven for thinking that RDF was the only possible representation for an OWL ontology. In part this is due to APIs and popular tools such as Jena[8], Protégé 3.X [7] and more recently TopBraid composer[9] whose 'data models' for representing OWL are entirely based on RDF. Such tools may give the impression that editing an OWL ontology *necessarily* involves editing an RDF graph. However, this is generally not the case and is reflected in the original design of the WonderWeb API, which was not biased towards any particular concrete representation. This has been carried though to the new OWL API .

In general, mapping to API interfaces and implementation objects is carried out at load time by parsers which conform to an `OWLOntologyParser` interface. A parser registry design makes it particularly easy to add parsers for new or custom syntaxes. At the time of writing, parsers for the OWL 1.1 Functional Syntax, OWL 1.1 XML, RDF/XML, KRSS, The Manchester OWL Syntax [8]

[8] http://jena.sourceforge.net

[9] http://www.topbraidcomposer.com

and the OBO flat file format[10] are bundled with the API. Parsers are selected automatically at runtime, which means it is possible to mix and match different concrete representations within a set of ontologies, for example within an imports closure.

Loading ontologies from documents is only one possible method of obtaining a concrete representation of an ontology. The design of the API makes is possible to provide mix and match implementations of the `OWLOntology` interface. For example, it is envisaged that an implementation could wrap some kind of triple store, custom database back end or remote ontology server. As far as possible, the API intefaces have been designed to support such implementations. For example, the API has been designed so that *immutable* axioms are the basic unit of currency. In order to manipulate an ontology, axioms are either added to, or removed from the ontology. While ontologies contain axioms, axioms don't belong to a particular ontology. The fact that axioms, and many other objects such as class descriptions, are immutable, and the fact that they do not hold references to ontologies, means that it is easy to make these objects, and therefore an ontology, persistent using a storage mechanism such as a database.

3.3 Support for modularity and multiple ontologies

One of the basic premises of the semantic web is that anyone can make any statement about any resource. This is reflected in OWL, where statements about entities/resources may be spread thoughout multiple ontologies. A key feature of the WonderWeb API, which was built in from the very beginning, was support for the manipulating and management of multiple ontologies. The new OWL API inherits the same design, making it possible to determine if an axiom belongs to a particular ontology. It is also possible to examine axioms that define an entity which reside in a particular ontology. For example, for a given class it is possible to obtain the super classes which have been asserted in a particular ontology, filtering out all other subclass axioms that relate to that class which have been asserted in other ontologies.

Ontology Imports A thorny issue in the development of many OWL tools is that of `owl:imports` – i.e. ontology inclusion. The original OWL 1.0 specification was incredibly unclear in this area. The notion of what should be at the end of an `owl:imports` triple – the object of the triple – is particularly fuzzy. Some developers have interpreted this to simply be a URL which points to a concrete representation of an ontology contained within some document. The approach taken by the OWL API is that an imports statement points to an ontology that has a *name* (URI) which corresponds to the object of the imports statement. In common with other APIs the OWL API uses a mapping mechanism, which takes an ontology URI and maps this to a physcial URI. This physical URI is finally resolved to point to some concrete representation of an ontology. From a

[10] http://www.geneontology.org/GO.format.shtml – note that a mapping define at http://www.cs.man.ac.uk/ horrocks/obo/syntax.html is used.

conceptual point of view it should be noted that this *is not* merely a redirection mechansim in the traditional sense of the web – i.e. from one physical location to another. The OWL API simply views ontology URIs as ontology names which don't contain any information about the physical location of the ontology. An analogy is that of imports in Java. When a programmer uses an imports statement in Java they are not specifying a physical location of a class file, they are specifying the *name* of a class. While it is typically the case that the physical location of a class file is related to the package name[11], this need not be the case - a class could originate from other sources such as a network, or may even be constructed dynamically. In other words, the particular implementation of a class loader determines how a class is loaded at runtime. In the case of the OWL API the equivalent of a Java class loader is the combination of an 'ontology URI mapper' and an 'ontology factory'.

In addition to the entire loading of all ontologies in an imports closure, the API supports the possibility of loading imported ontologies on demand and is tolerant towards missing imports[12]. In such a situation any reasoning will most likely be incomplete and possibly incorrect. However, import on demand can be incredibly useful in the context of making local changes and incrementally exploring ontologies in editing environments.

3.4 Reasoning

Reasoning tends to be one of the key features of ontology-based applications and tools. In common with the WonderWeb API, the OWL API provides general reasoner interfaces to describe different reasoner functionality, ranging from consistency checkers through to class based reasoning. At the time that the WonderWeb API was published, an emphasis on TBox reasoning was reflected in the design of the reasoner interfaces. However, with the development of modern reasoners such as Pellet and experience of developing applications as in [9] (exploring social networks), the ability to perform, and requirement for, ABox reasoning has come to the fore. Therefore, the OWL API has enhanced the reasoner interfaces for querying individuals. For example, to retrieve all individuals which are related via a given property, all property fillers for a given individual and property, most specific (or all) classes a given individual is instance of, etc. While the core API does not incorporate a complete query language, it tries to make a compromise between common query tasks and a pure query language, taking inspiration from the DIG 2.0 interface and various reasoners such as Pellet [10], FaCT++ [11] and Racer [12].

The general reasoner interfaces that the API provides make it possible for implementors to wrap their reasoners and provide OWL 1.1 API compatible implementations, thereby making them available for use by other OWL API tools or services such as black-box debugging as described later in section 3.6. No

[11] i.e. class path plus package

[12] Imports that cannot be loaded because a concrete representation cannot be obtained for example.

assumptions are made about the nature of the communication that takes place between the API and a reasoner implementation. However, when communicating with an external reasoner, for example via HTTP (in the case of DIG 1.1) or TCP, the reasoning process frequently takes less time than the serialisation, parsing and transport of messages involved in communication. In reasoner intensive applications this is usually unacceptable and it is perhaps for this reason that there has been a trend towards direct in-memory reasoner implementations. To date, such implementations have been provided by Pellet and FaCT++. These implementations are available directly from the Pellet website[13] and the FaCT++ website[14].

3.5 Support for change

The API uses the well known *Command* design pattern - all ontology changes are applied through change objects. Amongst other benefits this makes it possible to log/store changes, provide undo/redo support and transmit changes in client/server applications. The WonderWeb API used this approach with great success. In total there were around fifty different types of change objects. With moving to an axiom oriented API this has essentially been reduced to two change objects – `AddAxiom` and `RemoveAxiom`. The axiom that a change encapsulates provides the context for the type of change. All changes are now applied through an `OWLOntologyManager`, which has built in support for change history, change set annotation and undo/rollback.

3.6 Tools support

In the same way that the original WonderWeb API distribution bundled together several tools such as an OWL Validator and a DIG client implementation, the OWL API also bundles together several useful tools. Not only do these tools offer examples of API usage, they also provide 'out of the box' tools support for use in editors and similar applications. This section gives a brief example of such tools, for further examples and implementations please see the OWL API pages on Source Forge.

A Black Box OWL Debugger One of the most exciting tools bundled with the API is an implementation of a Black Box OWL Debugger. The implementation is based on the algorithms developed by Kalyanpur [6]. The debugger is capable of detemining the axioms responsible for an entailment in an ontology. For example the axioms that cause a class to be inconsistent, or the axioms which are responsible for a subsumption relationship between classes. The debugger is black-box in that it can be used with any sound and complete DL reasoner without any modification or tuning of the reasoner.

[13] pellet.owldl.com

[14] owl.man.ac.uk/factplusplus

Fragment detector and expressivity checker As part of the OWL 1.1 specification several tractable fragments have been identified. The API includes a fragment detector to determine if an ontology, or indeed a set of axioms, belong to one of these fragments. Additionally, a DL expressivity checker is provided, which determines the particular Description Logic that an ontology corresponds to.

4 Conclusions

An OWL API that supports OWL 1.1 is available for use in applications and tools that require OWL 1.1. The API closely follows the OWL 1.1 specification. In particular, the API subscribes to the axiom centric view of an ontology. The API is immediately available for download from the Source Forge Web Site: http://sourceforge.net/projects/owlapi.

Acknowledgements

This work was completed as part of the CO-ODE project funded by the UK Joint Information Services Committee (JISC). The authors would like to thank Evren Sirin from Clarke and Parsia for his valuable input and feedback during the development of the the new OWL API .

References

1. Liebig, T., Luther, M., Noppens, O., Paolucci, M., Wagner, M.: Building Applications and Tools for OWL – Experiences and Suggestions . In: Proc. of the OWL Experiences and Directions Workshop (OWLED'05) at the ISWC'05. (2005)
2. Bechhofer, S., Lord, P., Volz, R.: Cooking the Semantic Web with the OWL API. In: Proc. of the 2th International Semantic Web Conference (ISWC 2003). (2003)
3. Horridge, M., Tsarkov, D., Redmond, T.: Supporting Early Adoption of OWL 1.1 with Protege-OWL and FaCT++ . In: Proc. of the OWL Experiences and Directions Workshop (OWLED'06) at the ISWC'06. (2006)
4. Kalyanpur, A., Parsia, B., Sirin, E., Grau, B.C., Hendler, J.: Swoop: A Web Ontology Editing Browser. Journal of Web Semantics **4**(2) (2006)
5. Liebig, T., Noppens, O.: OntoTrack: A semantic approach for ontology authoring. Journal of Web Semantics **3**(2) (2005) 116 – 131
6. Aditya Kalyanpur, Bijan Parsia, E.S.: Black box techniques for debugging unsatisfiable concepts. In: International Workshop on Description Logics - DL2005. (2005)
7. Knublauch, H., Musen, M.A., Rector, A.L.: Editing Description Logic Ontologies with the Protégé-OWL Plugin. In: International Workshop on Description Logics - DL2004, Whistler, BC, Canada (2004) (2004)
8. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.H.: The Manchester OWL Syntax . In: Proc. of the OWL Experiences and Directions Workshop (OWLED'06) at the ISWC'06. (2006)

9. Noppens, O., Liebig, T.: Interactive Visualization of Large OWL Instance Sets. In: Proc. of the Third Int. Semantic Web User Interaction Workshop (SWUI 2006), Athens, GA, USA (2006)
10. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL Reasoner. Journal of Web Semantics (2006)
11. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006). Volume 4130 of Lecture Notes in Artificial Intelligence., Springer (2006) 292–297
12. Haarslev, V., Möller, R.: Racer: A Core Inference Engine for the Semantic Web. (2003) 27–36