Supporting Early Adoption of OWL 1.1 with Protégé-OWL and FaCT++

Matthew Horridge[1] and Dmitry Tsarkov[1] and Timothy Redmond[2]

[1] The University of Manchester, Manchester, UK
[2] Stanford University, Stanford CA, USA
{matthew.horridge|tsarkov}@cs.man.ac.uk, tredmond@stanford.edu

Abstract. This paper describes integrated tools support for OWL 1.1 in the form of the FaCT++ Description Logic reasoner and the Protégé-OWL ontology editor. Challenges of designing and implementing OWL 1.1 reasoning algorithms are highlighted, and an outline of an OWL 1.1 API and editing environment is provided.

1 Introduction

In addition to being a World Wide Web Consortium Standard, the success of OWL can perhaps be attributed to availability of tools such as Protégé-OWL [6], which facilitates the browsing and editing of OWL ontologies, and reasoners such as FaCT++ [9], which provides reasoning support for the Description Logic that underpins OWL. The provision of these tools ensured that both ontology developers and domain experts had easy access the powerful features of OWL for building ontologies and using them in end user applications.

However, at the OWL Experiences and Directions Workshop held in Galway in November 2005, the limitations with the then current version of OWL, herein referred to as OWL 1.0, were highlighted. These limitations were emphasised by OWL users, who found them to be an impediment to building OWL based applications. In particular, lack of qualified cardinality restrictions, expressive datatype reasoning, and property chain inclusion axioms proved to be major issues for the user community. In order to remedy this, an extension to OWL, called OWL 1.1, was planned. It was decided that the OWL 1.1 feature set would be just large enough to encompass the changes necessary to solve the immediate OWL 1.0 limitations, and crucially, the major reasoner implementors and tools providers committed to supporting this extension.

Ontology developers and domain experts are now looking forward to the langauge features offered by OWL 1.1. In order to satisfy the needs of these early adopters, and also ensure the wider uptake of OWL 1.1, an integrated toolset was produced: FaCT++ was extended to provide reasoning support and a new version of Protégé-OWL was produced to support the editing of OWL 1.1 ontologies.

The development of this OWL 1.1 toolset can be divided into three parts: (a) The design and implementation of the reasoning algorithms required to give

FaCT++ an OWL 1.1 reasoning capability; (b) the design and implementation of a communication interface to sit between the WonderWeb OWL API reasoner interfaces and the FaCT++ implementation – necessary while the DIG 2.0 specification was in flux; (c) the implementation of a new version of Protégé-OWL for editing OWL 1.1 ontologies, which was based on the WonderWeb OWL API. An overview of the architecture is depicted in Figure 1, and the suite of tools is described throughout the rest of this paper.

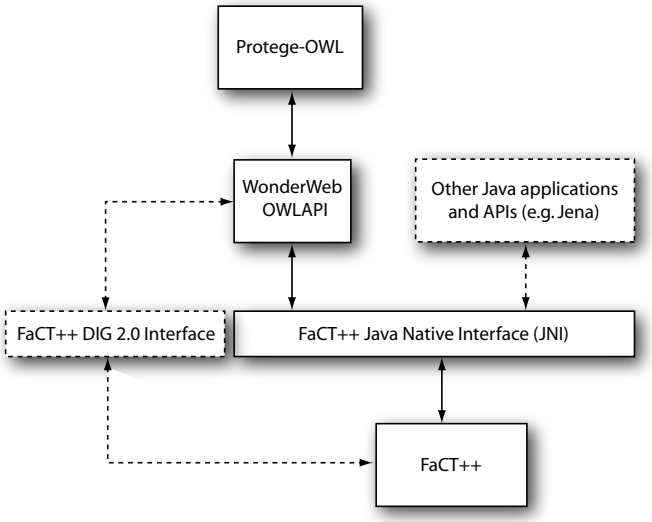

Fig. 1. An overview of the OWL 1.1 FaCT++ and Protégé-OWL toolset architecture. (Dotted lines and boxes show future work or potential configurations.)

2 Design and Implementation of Reasoning Algorithms

In order to provide reasoning support for OWL 1.1, the FaCT++ reasoner was extended. These extensions ranged from being trivial to implement through to extensions whose theoretical underpinnings require further research so as to optimise them.

The burden was lessened to some extent, since FaCT++ already supported some of the OWL 1.1 specification. For example, with the introduction of OWL 1.1, the requirement of the separation of names for classes, properties and individuals was lifted. This meant that the same name could be used as any or all of a class, property or individual. This mechanism, which is also known as *punning*, was already used by FaCT++. Additionally, although OWL 1.0 corresponds to

the $\mathcal{SHOIN}$ description logic, FaCT++ supports sound and complete reasoning for $\mathcal{SHOIQ}$– i.e. OWL 1.0 plus Qualified Cardinality Restrictions (QCRs).[3]

The OWL 1.1 extensions that were therefore required are summarised in the list below and are briefly discussed in the remainder of this section.[4]

- Local reflexivity ($\exists$ R.Self)
- Disjoint properties
- Reflexive, irreflexive and anti-symmetric properties
- Property chain inclusion axioms
- User derived datatypes

Local reflexivity was relatively easy to implement – for any individual x that must have a relationship along a reflexive property, an appropriately labelled edge $< x, x >$ is added to the model.

The processing of disjoint properties was split into static and dynamic parts. A dynamic analysis is applied to nodes that are merged during the reasoning process, and all other cases are handled by static analysis.

Support for reflexive properties was provided by introducing a new type of general axiom that is applied to all newly created nodes. In essence, this was similar to the approach taken to support local reflexivity, and was relatively straight forward.

Implementation of the algorithms for irreflexive and anti-symmetric properties is based on the previously implemented extensions for local reflexivity and disjoint properties, and essentially "came for free".

The vast majority of time was spent in providing reasoning support for property chains, which were considered to be the most challenging aspect. In order to maintain an acceptable level of reasoning performance, it was necessary to implement several optimisations. Discussion of these optimisations is beyond the scope of this paper, and will be dealt with in future papers.

Finally, datatypes and datatype reasoning have long been deemed to be very important by user communities. However, they are frequently seen as a boring distraction by the logicians. Fortunately, the importance of user derived datatypes and more expressive datatype reasoning has begun to be addressed with OWL 1.1, and FaCT++ now supports a new datatype reasoner architecture, which is extensible and has the capability of being pluggable.

2.1 Communicating with FaCT++

Since its inception, FaCT++ has supported the DIG reasoner communication protocol. DIG [3] is a lightweight XML over HTTP based protocol that allows an ontology to be fed into a reasoner (using *TELL* statements) and then queried

[3] Historically, this is due to the reason that the original FaCT reasoner had support for QCRs, and DIG 1.1 also supports QCRs.

[4] An in-depth discussion of algorithm design and optimisation is beyond the scope this paper. A future paper will address the issues of algorithm design and optimisation in detail.

(using *ASK* statements). Queries often take the form of simple questions about the structure of the ontology, such as queries about subsumption relationships between class descriptions. The main benefit of DIG is that any DIG compliant reasoner can be interchanged with another DIG compliant reasoner. Indeed, in previous versions of Protégé-OWL, the main channel of communication with DL reasoners was via the DIG protocol, thus encouraging this interchangeability. However, the current version of DIG (DIG 1.1) only supports a subset of OWL 1.1. In order to address this, a working group was set up to produce the next version of the DIG standard – DIG 2.0[2]. One of the goals of the working group is to ensure that DIG 2.0 can support the representation of OWL 1.1 ontologies.

At the time of writing, the DIG 2.0 specification has not been finalised and is still under active development. In order to make OWL 1.1 features available to the community as soon as possible, it was decided to produce a direct bridge to FaCT++, and provide DIG 2.0 support as soon the specification is completed. Since the vast majority of OWL ontology tools are written in Java, and FaCT++ is written in C++, the interface to FaCT++ was implemented using the Java Native Interface (JNI). Inspiration for the design of the interface was taken from the DIG specification. A number of API functions were implemented that roughly correspond to the DIG concept/role language, DIG axiom (*TELL*) syntax and the DIG query (*ASK*) language, in turn these roughly correspond to the OWL 1.1 functional style syntax [8]. The benefit of this approach is that any Java ontology API, such as the Protégé-OWL API [7], WonderWeb API [1] or the Jena API[4], could potentially be extended with a communication layer in order to interact with FaCT++ while the DIG 2.0 standard is being realised.

3 Implementation of an Ontology Editing Environment

As mentioned previously, one of the factors that undoubtedly placed OWL in the category of most widely used ontology languages was the availability of free editing and browsing tools such as Protégé-OWL or SWOOP [5]. Support for the editing and presenting OWL 1.1 ontologies is therefore hugely important. To this end, it was decided that the popular ontology editor, Protégé-OWL would be used as the basis for an OWL 1.1 editor.

To provide editing and ontology manipulation support, it was necessary to create an OWL 1.1 API. The obvious route forward was to take an existing API and extend it to support OWL 1.1 syntax and objects. Although Protégé-OWL had its own OWL API, it had evolved over time and had become bloated, with the mixing of OWL with RDF(S) and presentational information with data model representation – hence, extension of the Protégé-OWL API was ruled out forthwith. The other main OWL APIs to choose from, were the WonderWeb OWL API and the Jena API. Since Jena is based on RDF triples, and an OWL 1.1 mapping to RDF graphs (if possible) has yet to be decided upon, the WonderWeb OWL API was the API of choice.

An attractive feature of the WonderWeb API is that it is not tied to any particular concrete representation of OWL. This meant that the modification of

the API to support OWL 1.1 constructs was a very straight forward process. Issues such as representing OWL 1.1 constructs using triples could be ignored.

One of the potential issues that developers face when building OWL 1.1 support into existing OWL APIs and tools is the issue of punning. With the existing Protégé-OWL APIs, and perhaps some RDF(S) based APIs, it would have been very difficult to cope with punning without a major restructuring of the underlying implementation. Happily, the WonderWeb API already had direct support for punning, which mean it was a non-issue.

The existing version of Protégé-OWL was built as a plugin to the core system. This meant that there was a layering on top of the original frames system. In addition to this frames layering, there was an intermediate RDF(S) representation. Any calls to the API to modify an OWL ontology were translated to calls to modify an RDF graph, which were translated to calls to modify frame-slot-value-facet quads. When the requirement to support OWL 1.1 arose, it was decided that the optimal solution would be to reimplement Protégé-OWL using the WonderWeb OWL API. The benefits of this were twofold: (a) The efficiency and robustness of the system, in terms of loading, and manipulating ontologies improved dramatically; (b) dealing with a "native" OWL API proved to be much cleaner, divorcing the issues of concrete representation from programmatic manipulation of an ontology. In particular, because the existing version of Protégé-OWL was tied to a triple representation, it would have been difficult to support the annotation of axioms and other arbitrary comments that are allowed in OWL 1.1. In terms of the Protégé-OWL GUI, the look and feel of many components was kept, and only minor extensions were required to support OWL 1.1 editing. Generally speaking, the widgets for displaying and editing properties needed extending to cope with disjoint properties, the additional property characteristics and property chain inclusion axioms.

4 Conclusions

- An integrated set of OWL 1.1 tools has been developed, in the form of FaCT++ and Protégé-OWL, in order to encourage the uptake of OWL 1.1. The tools will enable users to access the critical features OWL 1.1 that are necessary for many OWL based applications and were previously missing from OWL 1.0.
- When implementing OWL 1.1 reasoning algorithms, once algorithms for disjoint properties, and reflexive properties had been implemented, support for irreflexive properties and anti-symmetric properties essentially "came for free".
- The most challenging aspect of implementing an OWL 1.1 reasoner, was the implementation of *property chain inclusion axioms*. Several optimisations were necessary to ensure acceptable reasoner performance.
- The Protégé-OWL ontology editor was re-implemented to sit on top of the WonderWeb OWL API which was extended to support OWL 1.1. This ensured that the representation of OWL 1.1 ontologies was not dependent on

a particular representation such as RDF triples. Moreover, it made support for punning and arbitrary comments/annotations possible – something that would have been difficult to accomplish using the existing Protégé-OWL triples based representation.
– The OWL 1.1 specification was designed to be a small step in the right direction. In terms of extending existing OWL 1.0 reasoners and editing tools to cope with OWL 1.1, it has been found that the effort required to implement such extensions was perfectly acceptable, and minimal in comparison to the task of developing such tools from scratch.

5 Future Work

An outstanding, and *major* issue, is that there is currently no support in the modified OWL API or Protégé-OWL for a text based serialisation of OWL 1.1 ontologies. This is due to the fact that a triple representation, or an XML based representation in the flavour of the OWL XML Presentation Syntax, hasn't been finalised. In the mean time, ontologies can be saved in a binary format, using the Java object serialisation mechanism.

The authors are committed to implementing "native" DIG 2.0 support in FaCT++ and Protégé-OWL. This work will be completed as soon as possible after the DIG 2.0 standard has been finalised.

Acknowledgements

This work was supported in part by the CO-ODE project funded by the UK Joint Information Services Committee. Dmitry Tsarkov is funded by the Sealife project (IST-2006-027269). Special thanks to all at Stanford Medical Informatics for their continued collaboration – in particular Tania Tudorache. Special thanks also to Sean Bechhofer from the Information Management Group at the University of Manchester for help and advice on the WonderWeb OWLAPI.

References

1. S. Bechhofer, R. Volz, and P. Lord. Cooking the Semantic Web with the OWL API. In *Proc. of the International Semantic Web Conference (ISWC-03)*, 2003.
2. Sean Bechhofer, Thorsten Liebig, Marko Luther, Olaf Noppens, Peter Patel-Schneider, Boontawee Suntisrivaraporn, Anni-Yasmin Turhan, and Timo Weithoner. DIG 2.0 — towards a flexible interface for description logic reasoners. 2006.
3. Sean Bechhofer, Ralf Moller, and Peter Crowther. The DIG description logic interface. In *Proc. of the International Workshop on Description Logics (DL2003)*, 2003.
4. Jeremy J Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: Implementing the semantic web recommendations. Technical report, HP Labs, 2003.

5. Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. *The International Journal on Semantic Web and Information Systems*, 1(1), Jan–Mar 2005.
6. H. Knublauch, M. A. Musen, and A. L. Rector. Editing description logic ontologies with the Protégé-OWL plugin. In *Proc. of the International Workshop on Description Logics - DL2004*, 2004.
7. Holger Knublauch and Matthew Horridge. The Protégé-OWL API. `http://protege.stanford.edu/plugins/owl/api/index.html`, 2005.
8. Peter F. Patel-Schneider. OWL 1.1 web ontology language functional style syntax. http://owl1-1.cs.manchester.ac.uk/syntax.html, 2006.
9. Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR 2006)*, 2006.