Something Nasty in the Woodshed: The Public Knowledge Model

Robert Stevens[1] and Phillip Lord[2] and Andrew Gibson[1]

[1] School of Computer Science, University of Manchester, Oxford Road, Manchester, UK, M13 9pl
`robert.stevens@manchester.ac.uk`
[2] School of Computing Science, Newcastle University, Claremont Tower, Newcastle-Upon-Tyne, UK `phillip.lord@newcastle.ac.uk`

Abstract. Ontologies encoded in OWL-DL can be complex and even arcane to all but the encoders themselves. It is well understood that the user model or user view of this knowledge model should be different. The ontology knowledge model might, for instance, drive a forms based interface populated with appropriate terms driven by knowledge captured in the ontology. The knowledge model, however, will be used as a component via some API and the application is itself a user. There are often parts of the knowledge model that, while being a representation of the universe being modeled, do not or should not be seen by the application, let alone the end-user of, for instance, the forms. In this position paper we speculate on requirements for metadata that would make such knowledge artefacts "invisible" to the user, but remain part of the knowledge model that is reasoned over. We also discuss possible options and consequences for hiding parts of OWL-DL encoded ontologies.

1 Introduction

OWL-DL otologies can be very complex artefacts. The separation of the model from the users' view of that model is standard architectural practice, in many areas of computer science. A biologist using a terminology delivered by an OWL ontology should neither need to nor should be required to see the complexities of the underlying OWL model. In this position paper, we suggest the requirement to support an application view of the knowledge model, separate from the ontology builders view, to better enable presentation to the end user.

An OWL ontologist will use, by necessity, many knowledge components within the ontology that are required to achieve a high-fidelity representation of the domain. The most obvious example might be the upper level ontology needed to make ontological distinctions. Ontology design patterns, used to extend the capabilities of the language, often include components necessary to enable the pattern to achieve its goal. It is not only the end-user that needs not to see this portion of the knowledge model, it might also be hidden from the direct user of that knowledge model, such as an application builder.

The GRAIL language provided a feature for this purpose [1]. The keyword `invisible` was part of the language itself and could be applied to concepts and their criteria in an ontology [1]. These concepts did not appear in any application driven via the GRAIL terminology server (TeS). The *invisible* components of the ontology were nevertheless still present as part of the knowledge server and could be queried via the TeS and were reasoned over with the rest of the ontology; but those *invisible* components were not available outside the TeS, therefore providing the separation of knowledge model and application view.

In this paper, we are not arguing for extensions to the formal semantics of OWL. As with GRAIL, invisible components would be still be available to the reasoning process. In the rest of the paper for which components could be rendered invisible, and offer a brief discussion of mechanisms for achieving this goal.

2 What do we want to do

The ComparaGRID project[3] uses an application ontology that primarily provides a controlled vocabulary to facilitate database integration. Rather than directly engineering an ontology to be used by the ComparaGRID application, the project employs a strategy of engineering a much richer domain ontology for comparative genomics, from which the more specific application ontology can be derived at any time. The ontology covers: biological sequence features; maps; evidence for findings; and biological entities such as chromosomes. After ensuring that the initial domain ontology satisfies all of the basic requirements of the application developers, the domain ontology can be “trimmed down” to an application view of that ontology.

Currently, the process of “trimming down” the domain ontology into an application ontology is not efficient. Typically, this will involve removing some of the artefacts of knowledge engineering, such as some highly detailed leaf ontology terms used in the design process.

2.1 Knowledge Model Presentation Requirements

In this section, we describe the components of the knowledge model that we would wish to make *invisible* in producing an application view of the knowledge model or, specifically, the application ontology of ComparaGRID:

1. Hide class x. For example, if z is part of y, then x might be created as a superclass of z, but serves no other purpose than grouping it’s children. After hiding x, children to x would have to be made to appear to be children of all superclasses of x, except where this resulted in entirely redundant subclass relationships.
2. Hide all asserted descendants of x_i; we may wish to reduce the complexity (and the specificity) of the ontology. This should be simple—we just appear to have a smaller ontology.
3. Hide all asserted ancestors of x; for example, while upper ontologies or even owl:Thing may increase interoperability, they, often, do not need to be seen by the application view. This also appears straightforward.

[3] `http://www.comparagrid.org`

4. Hide a specific layer or interval of classes. Again this is useful for reducing complexity. As this generalises hiding a class, it should involve no additional complexity. We are aware that the concept of level is somewhat undefined in any ontology hierarchy excepting for trees. In the first instance, we suspect that level can be best interpreted as any class between these the two specified classes by any route.
5. Hide this restriction. For example, we have achieved disjointness between large numbers of siblings, using functional datatype properties to ordinal numbers, to avoid quadratic disjoint statement explosions[4]. While the use of OWL is still immature, such “dirty tricks” are inevitable, but should definitely be hidden from the end application as they are not part of the knowledge domain.
6. Hide this property. This would be useful in a complex hierarchy of properties created to achieve the desired effects from reasoning. A restriction using an invisible property would be shown to use the first visible super-property; we consider that hiding root properties would be inappropriate.
7. Hide this axiom. Again considering large numbers of disjoint siblings—even in OWL 1.1—a large number of disjoint statements for each class, is likely to be unwieldy in any application view. Rendering these invisible, also appears to be straightforward.
8. Hide an imported OWL module. This seems equivalent to hiding a number of individual classes.

3 Mechanisms

There appear to be a series of possible mechanisms for fulfilling these requirements for specifying invisibility.

First, it would be possible to extend the formal semantics of OWL; while this might produce the most elegant and expressive solution, there is no formal theory and would take a significant investment of effort.

Second, we could use the existing semantics of OWL; for example, we might define an `Invisible` class all of whose children would be so. This would enable us to state, for example, the general properties of a class which make in invisible. However, this mechanism would not extend readily to properties. It also conflates the knowledge model with the delivery of that model, which is ontologically undesirable[5] and would increase the computational requirements for reasoning.

Third, a construct in a macro expansion language could be used in the application view. The knowledge model could then be generated by macro expansion, which therefore fulfils the requirements of separating the two. We see issues with this. Currently no macro language exists and would require extensions to the OWL specification. Also a macro language could also be used for many other purposes therefore occluding the notion of invisibility.

Finally, and we believe most plausibly, a defined set of annotation properties could be used to provide information about the visibility of most of the various entities described above. Additionally, the ability to express ”always visible” may be necessary,

[4] We are aware that this problem should be more elegantly soluble in OWL 1.1

[5] Of course, the Invisible class would be a subclass of itself, and so would be invisible

to address the case where a class which we wish to be visible is inferred to be part of a layer marked as invisible. There may be a need to discriminate between behaviour of invisibility in the asserted and inferred hierarchy—this would certainly appear to be the case for direct children or parents as this is only clear within the asserted hierarchy.

4 Discussion

Here we have discussed the requirements and possible mechanisms to facilitate the separation of an application view from a possible complex knowledge model. This is a feature that has proven useful in the past. We suggest this as a light-weight option rather than a full-blown mechanism for enabling ontology views. Similarly, while it has some features that might be desirable for implementing a modularity mechanism, it would only be a small part of one.

Obviously, such a mechanism as described here could be dangerous if overused: the application view of the ontology would be different from the underlying knowledge model; this could have consequences unexpected by the user—searches might return apparently incomplete results. Such a feature should only ever be used sparingly, to hide those components of the ontology which would be genuinely inconvenient in the application view. OWL-DL is a powerful language and many of its features can have wide-ranging consequences in an ontology; like these features, that a visibility mechanism could, at times, have undesirable consequences does not bar it from having utility. We would recommend any implementation had an option for bypassing this feature.

Despite these caveats, as we are experienced at building ontologies which are required for applications within the life sciences, we believe that such a mechanism would be useful. As ontology builders, rather than language experts, we welcome interaction with the latter community to ensure that fulfilling these requirements is possible.

Acknowledgements: Andrew Gibson is funded by BBSRC grant BBS/B/17156.

References

1. A.L. Rector, Sean Bechhofer, Carole Goble, Ian Horrocks, W.A. Nowlan, and W.D. Solomon. The GRAIL Concept Modelling Language for Medical Terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.