The Protégé OWL Experience

Holger Knublauch[1,2], Matthew Horridge[1], Mark Musen[2], Alan Rector[1],
Robert Stevens[1], Nick Drummond[1], Phil Lord[1],
Natalya F. Noy[2], Julian Seidenberg[1], Hai Wang[1]

[1] School of Computer Science, the University of Manchester, UK
[2] Stanford Medical Informatics, Stanford University, CA, USA

holger@knublauch.com

Abstract. Protégé is one of the most widely used development platforms for ontology-based systems. We report on our experiences with the development of OWL support for Protégé, and on the experiences of our user community with OWL. While the overall feedback from the community has been positive, our experience suggests that there are considerable gaps between the user requirements, the expressivity of OWL, and users' understanding of OWL. In this document we walk through a selection of these issues and suggest directions for future work and standardization efforts.

1 Introduction

An ontology development tool is often the first thing that people get to see when they venture into the Semantic Web field. Ontology editors and visualization tools therefore carry a special responsibility for the success of the Semantic Web community. At the same time, the user communities around such tools serve as melting pots which can be exploited to collect feedback on the overall design of the language and associated systems.

Protégé has been a leading ontology development tool for more than a decade. While the traditional architecture of Protégé is based on frames [2], our team has extended it with comprehensive support for OWL [1]. The current Protégé version[1] can be used to edit classes and their characteristics, to access reasoning engines, to edit and execute queries and rules, to compare ontology versions, to visualize relationships between concepts, and to acquire instances using a configurable graphical user interface. Figure 1 shows a screenshot of Protégé's OWL class editor. Protégé is also an open platform into which arbitrary services can be added. For this purpose, Protégé provides a comprehensive Java API for working with OWL and RDF models. This API can also be used to develop stand-alone Semantic Web applications.

Being one of the most complete OWL editing tools, Protégé-OWL has been eagerly embraced by many in the user community since its first prototypes in 2003. The tool has evolved rapidly in response to feedback from its users. Furthermore, it has

[1] This document refers to Protégé 3.1, which is available from http://protege.stanford.edu

been greatly extended by project partners and external groups. While it is difficult to estimate the number of active Protégé-OWL users, the OWL mailing list alone has currently more than 1300 members. While many of these users have migrated to OWL from frame-based Protégé modes, there is also a large fraction of new users who are attracted by the Semantic Web vision.

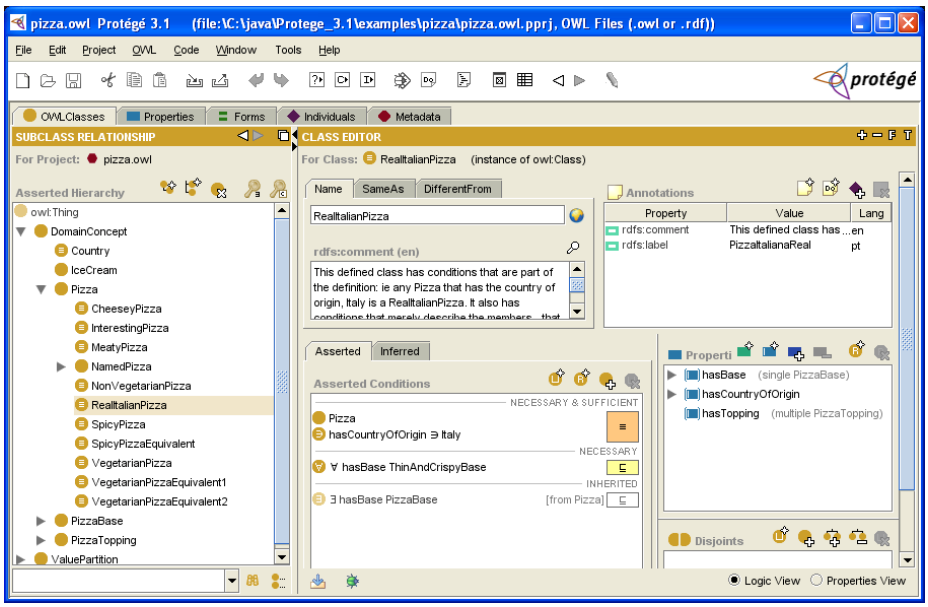

Figure 1: A screenshot of the class editor of Protégé-OWL. The hierarchy of classes is shown on the left and details of the selected class on the right. These details include annotations, properties and logical conditions of the class.

Another major reason why users have discovered Protégé is that OWL has become an official W3C recommendation. This status as a standard language not only encourages users to start building and sharing models, but it also means that a larger repository of reusable software components is available. For example, the availability of the Jena library[2] enabled rapid progress in the development of Protégé-OWL because it was possible to reuse a well-tested parser instead of implementing our own. Later, we could reuse reasoning services from the WonderWeb OWL API. Finally, we were able to seamlessly integrate a Jena-based SPARQL query engine into Protégé-OWL, because both Jena and Protégé map their internal storage models into a standardized RDF triple store. In the same way that we have benefited from third party components, many projects have adopted the Protégé-OWL API as the base platform of their own application. The finalization of OWL as an official W3C recommendation means that developers now have a common target platform to gain interoperability. The time of "ontology language wars" in the 1990s has fortunately come to an end.

[2] Jena is an open-source Java API for RDF/OWL; http://www.hpl.hp.com/semweb/jena.htm

Other driving forces behind the success of Protégé seem to be its availability as open-source software, its extensibility and the benefits that come from an active support mailing list. This mailing list has also evolved into a general discussion forum for questions about modeling and reasoning techniques. We have acquired additional feedback during a series of tutorials and workshops on OWL using Protégé[3]. The following sections will report on experiences we have collected during the development of Protégé for OWL, and feedback from Protégé users and tutorial attendees. The first part of this paper (section 2) will comment on issues with the current state of OWL and its associated techniques. We will illustrate some common misunderstandings about OWL which indicate a shortfall in the availability of tutorials, design patterns and tool support. The second part of the paper (section 3) focuses on user requirements that are currently beyond the scope of OWL, suggesting potential extensions to the language specification in the future. We will summarize our findings in section 4.

2 Understanding and Using OWL

From our experience with OWL tutorials and from providing technical support to Protégé-OWL users and our collaborators, we identified a number of difficult issues in the language design that users often struggle with (see also [5]). Better tutorials and education efforts can address some of these issues; tool design can alleviate other issues.

2.1 From Object-Orientation to Description Logics

Learning a new language is always difficult, especially if a new paradigm such as description logics (DL) is involved. Here – one would guess – previous experience with similar languages should be beneficial in order to draw parallels. Most computing professionals who explore OWL will likely have prior knowledge of object-oriented languages such as UML. In the field of knowledge modeling, many people have background in frame-based systems. However, we argue that the OWL community must very carefully explain the differences between object-oriented approaches and DL to such people. There are many subtle but crucial differences between these paradigms, which often lead to misunderstandings by both users and developers.

Here are examples from our own experience: prior to starting the Protégé-OWL project none of the developers had extensive knowledge of DL, though all had considerable background in mainstream object-oriented techniques. However, both the model and user interface components of Protégé-OWL had to iterate through many designs. While early versions of the user interface in 2003 were essentially remakes of rather frame-based restriction editors, we later understood that the distinction between primitive and defined classes (also known as “complete” and “partial” defini-

[3] Much of the teaching material is available online from http://www.co-ode.org

tions) was rather awkward to model this way and opted for different style of representation which has evolved into the current "conditions widget" (that is shown in the center of Figure 1). Similarly instructive were our early attempts to map DL constructs such as `rdfs:domain` into their frame based counterparts: in most frame based systems domains are in effect mandatory and act as invariants that either produce errors or cannot be violated by nature of the interface. By contrast, in OWL, domain constraints are axioms from which inferences may be drawn. Furthermore, properties often have no domain statements at all, while object-oriented attributes must be attached to certain classes. Other sources of misunderstandings are the open-world assumption and the lack of the unique name assumption.

While the above problems were experienced in the design of Protégé-OWL itself, similar problems are likely to arise for other software systems. Until Semantic Web technology has been established as a standard skill of mainstream developers, the majority of developers will not have the "correct" understanding of OWL. We therefore argue that the OWL community should be very careful to explain the differences between these paradigms and seemingly related mainstream approaches. An instructive "OWL for UML users" document is clearly needed and should be widely distributed, especially through non-academic publication channels. The publications of the OMG Ontology Working Group[4] could serve as a helpful starting point.

2.2 OWL Species and Types of Users

Many Protégé users do not exploit (or require) the full range of constructs available in OWL. Our experience suggests that the majority of users regard OWL mostly as a more expressive variant of RDF Schema, i.e. they use it as a mechanism to define classes, properties and individuals in a format that can be shared on the Web. OWL greatly extends the expressivity of RDF Schema, and many users exploit restrictions to express what they perceive to be necessary conditions of a class, and use the `owl:imports` mechanism to link ontologies with each other. Such ontologies carry little semantics that could be exploited by reasoners. In particular, the distinction between defined and primitive classes is not fully understood by most users, and therefore few equivalent class definitions are used. Likewise the issue of open-world semantics and absence of the unique name assumption are often ignored by many ontology designers.

While many OWL DL supporters argue that the Semantic Web will not make sense without a clean logical foundation, there are valid use cases for exploiting only subsets of OWL. The selection of an OWL dialect depends on whether users primarily build taxonomies, data structures or rich knowledge models. For example, an e-commerce ontology may contain "data" classes describing customers together with their address and phone number. An initial version of such an ontology does not require advanced OWL constructs beyond range and domain statements. These semantically simple ontologies would be sufficient to drive a Web application that generates user interface forms from class definitions and to describe schema that can then be

[4] http://www.omg.org/ontology/

integrated with conventional data bases. At later stages of the ontology's life cycle, developers may find out that they need additional expressivity, for example to classify customers by their purchases. This is one of the major selling points for OWL that is often ignored by DL evangelists. The breadth of the OWL language offers a migration route from entry level, hand-crafted taxonomies of terms, to well defined, normalized ontologies capable of supporting reasoning.

In order to support the various subsets of OWL, editing tools such as Protégé face the risk of becoming too baroque and overwhelming for some users if they cannot be tailored to the specific expressivity required. We have implemented mechanisms to reduce the available features (e.g., a rather object-oriented view in addition to the DL-based logic view), but more work needs to be done to identify classes of users and to provide the appropriate level of abstraction for them. For example, a logician's user interface that displays using logic notations will not work for other communities. Many users wish to build ontologies on the large scale, so a user interface that really only suits small scale development for probing the capabilities of reasoners and language features will similarly fail to be adopted. Long experience of application development reveals that misunderstanding types of users can be fatal to software projects.

2.3 Namespaces, Imports and Syntactical Problems

The layering of OWL on top of RDF is perceived by most users as an advantage, because this layering is the base of modularizing and reusing ontologies on the Web. However, many users have trouble understanding the differences between an ontology's physical location, its default namespace, and its `xml:base`. One problem is that the physical location of an ontology can be different from the base location that is being defined inside the file. Furthermore, the default namespace of the file can be yet another URI. This is particularly confusing when users work with modular ontologies that import each other: on the Semantic Web, import statements should point to an http:-based address, but at edit-time, the imported file will be located in a local file folder. It is difficult for tools like Protégé to implement a comprehensive but transparent mechanism to redirect such imports to local files.

Another common error is the lack of import statements and the resulting lack of `rdf:type` statements for OWL DL semantics. While it is a common practice in RDF(S) ontologies to establish references between models by simply declaring and using namespaces, it is unclear to many OWL users that declaring a namespace for an external ontology is not sufficient to import it. Tools should provide pro-active support to detect typical beginner's mistakes such as missing imports.

In the context of imports and modules, the Semantic Web community could learn from mainstream methodologies. For example, object-oriented languages are based on encapsulation, making it possible to build models out of reusable components with well-defined interfaces. With OWL and RDF it is currently unclear how to exploit the namespace and import mechanisms to structure ontologies into public and private parts, or interface and implementation ontologies. Future versions of OWL should provide much stronger guidelines for building modular ontologies.

2.4 The XML Syntax of OWL and RDF

An obvious source of problems with RDF is its default XML syntax. A large fraction of users' questions comes from people whose existing OWL models fail to load into Protégé and generate error messages from the parser. On further inspection it often turns out that the users have manually edited the file on XML level, or produced the files with an external script. Our mailing list contains strong evidence that the XML serialization of OWL should almost be categorized as a binary format that should not be edited outside of specialized tools. This does, however, leave the question of imports from other tools and the converters to do so. Many of our users from the life sciences need to convert, for example, data produced from analyses into a form that can be loaded into OWL/RDF tools. Using an intermediate representation, such as the OWL abstract syntax, is one route for simplifying conversions. Another route is making parsers more robust and helpful to detect and perhaps fix typical errors.

3 OWL Language Extensions

OWL DL is based on what is regarded as a highly expressive description logic. This expressiveness is particularly beneficial to users in biomedical communities, who typically operate on deeply nested, tangled and multi-axial terminologies. However, there are a number of areas where the OWL language does not have the expressive power that our users require. In many cases, additional constructs can be introduced by the tool, but standardized solutions would improve interoperability.

3.1 Concrete Domains and User Defined Datatypes

One of the omissions in the OWL language that our users complain about most often is poor representation of numeric expressions. Almost all groups, except for those developing traditional medical terminologies, sorely need to be able to express quantitative information. Typical examples include the length between 1mm and 2mm, age greater than 18 years, pressure in the range of 1030mb to 1035mb. Such range declarations are needed to classify individuals and to build class definitions such as "Adult", and should therefore be supported by reasoners [3]. Our user base points out that the current OWL datatype formalism is much too weak to support most real world applications and that many potential users therefore cannot adopt OWL[5] [6] [7] [8].

The user communities anxiously await an extension to the OWL specification to represent user-defined datatypes with XML Schema facets such as `xsd:minInclusive`. We are beginning to support this in Protégé already with our

[5] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/4033
[6] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/5092
[7] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/7607
[8] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/11757

own encoding conventions, which we will convert once a standard becomes available. However, the current standardization draft[9] aims at a solution which would maintain datatype declarations outside of the OWL files, and linking the OWL file with the external datatype constructs by means of URI references. Based on our experiences with the problems in the imports mechanism we encourage the OWL community to produce a rendering of XML Schema datatypes as inline declarations. This approach would allow users to encapsulate datatype definitions inside of ranges and restrictions[10].

Even if reasoning with user-defined datatypes can not be fully supported by existing tools, the OWL specification should at least provide a standardized mechanism for expressing such constraints, for example to validate user input on forms.

3.2 Qualified Cardinality Restrictions

DAML+OIL, the predecessor language of OWL DL, had support for qualified cardinality restrictions (QCRs). Such restrictions allow class expressions to be formed that talk about the class of individuals that have a certain number of relationships along a given property to members of a *specific* class. For example, QCRs can declare the class of things that have two parts which are Legs.

QCRs were not included in the OWL specification. Instead, they were "downgraded" to plain cardinality restrictions, which allow simple expressions such as the "class of things which have two parts" to be formed. In essence, QCRs were regarded as being too complicated for users to understand and too difficult for tools such as Protégé to support[11]. We believe that this was a major omission in the OWL specification, since QCRs are required in many applications. Again, our main user community from the life sciences is rife with this requirement and its omission from OWL is a serious hindrance to its uptake by this community [7]. The Protégé discussion list also contains requests from our users[12] [13] [14] [15] [16]. While work-arounds have been suggested[17] these do not capture the full semantics of QCRs.

QCRs are well understood from a logical and reasoning point of view. Indeed one of the most popular DL reasoners, Racer, supports reasoning with QCRs and there are

---

[9] http://www.w3.org/TR/2005/WD-swbp-xsch-datatypes-20050427

[10] http://protege.stanford.edu/plugins/owl/xsp.html shows the current Protege-OWL implementation of user-defined datatypes

[11] http://www.w3.org/2001/sw/WebOnt/webont-issues.html#I3.2-Qualified-Restrictions The difficulties for Protégé were originally explained by the limitations of the facet mechanism in the frame-based knowledge model. However, since Protégé OWL now represents restrictions by means of RDF triples, the limitation no longer exists.

[12] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/8883

[13] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/4607

[14] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/9804

[15] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/3256

[16] http://article.gmane.org/gmane.comp.misc.ontology.protege.owl/4641

[17] http://www.cs.vu.nl/~guus/public/qcr.html

plans to add support into FaCT++. Contrary to the claims of the committee, in large numbers of tutorials our experience is that users find them intuitive, often more intuitive in general than simple cardinality constraints. To satisfy the demands from different user communities, Protégé-OWL already supports creating class descriptions using QCRs. The implemented solution expresses QCRs as cardinality restrictions with a third property (currently called `owl:valuesFrom`). The same convention is used by Racer. Whilst this solution goes beyond OWL DL, we believe that having this option was a necessary extension.

3.3 Possibilities and Optionality

OWL is based on two valued logic about what is universally true. It provides no mechanism for defeasible reasoning. It has no inbuilt support for reasoning about what is "typically" or "generally" true. In some cases, where there are a limited number of exceptions, this can be handled by making the logical statements more specific, *e.g.* by saying that "(all) Eurkaryotic cells *except* mammalian red blood cells have nuclei" instead of "(all) Eukaryotic cells have nuclei". In other cases, the patterns are more complex and this approach leads to combinatorial explosions [4].

Users also need to be able to say that things "may occur" – *e.g.* a drug "may have a side effect" – it does not always, but it may. Medical resources particularly require this type of representation as shown by experience with local experts.

Some users, particularly those used to working in UML, need to be able to say that a property is optional.

There are no standardized OWL language elements for expressing these types of possibilities and optionality. An in-depth discussion of the resulting problems and potential solutions is beyond the scope of this paper, but based on our experience we argue that these questions should be considered for future versions of the OWL specification.

3.4 Disjoint Classes

OWL DL has the `owl:AllDifferent` construct to make a set of individuals mutually distinct from each other. However, there is no corresponding construct to make a set of classes mutually disjoint from each other. Even though the OWL Guide[18] states that, "A common requirement is to define a class as the union of the set of mutually disjoint subclasses", it later says, "As the number of mutually disjoint classes grows the number of disjointness assertions grows proportionally to n^2. However, in the use cases we have seen, n is typically small". Our experience in practice is that this is not the case – n is typically large enough such that the number of disjoint axioms becomes seriously problematic. There are, for example, twenty amino acids, hundreds of protein domains and thousands of species; all are disjoint.

[18] http://www.w3.org/TR/2004/REC-owl-guide-20040210/#DisjointClasses

Researchers have suggested that every primitive open concept should be part of a disjointness axiom with its siblings [6]. When the number of sibling classes is small the implementation of this design pattern does not pose any problems. However, with an ontology the size of GALEN[19], where n is significant, the inefficiency of asserting pairwise disjoint axioms quickly becomes impractical.

For example, the GALEN "NamedActiveDrugIngredient" class has 1492 primitive subclasses (n=1492). Making these classes pairwise disjoint adds 2226064 triples to the ontology. To assert all siblings in GALEN mutually disjoint requires a total of around 2600000 triples to the ontology. In terms of file size, adding these triples causes the 20 MB OWL file to bloat to 200 MB. Similar problems are found in other biomedical ontologies such as SNOMED, the NCI Thesaurus and the Gene Ontology.

We recommend that an `owl:AllDisjoint` construct should be added to the OWL specification in order to ameliorate this situation. This would not only serve as "syntactic sugar" for OWL files, vastly reducing the file sizes, but would also bring OWL closer to the DIG[20] protocol which is used to communicate with DL reasoners.

3.5 Additional Standard Metadata

A great value of OWL is its extensibility. Users can easily define their own vocabularies and thus even define new languages on top of OWL. Such language extensions are often driven by groups such as the Web Services community that came up with OWL-S and the librarian community with its Dublin Core framework.

Similar extensions are likely to emerge from other communities. We anticipate that tool developers will come up with *de-facto* standard ontologies for various purposes. In particular, we found that it would be useful to have metadata about visual forms, such as whether a certain string property value should be edited in a multi-line text area or a simple text field, and whether the `firstName` property should be displayed to the left of the `lastName` property. Such information is currently stored in Protégé in a native format, but we expect to make this application independent in the future. Ideally, the same metadata could then also be exploited to generate Web forms and PDF documents. Moreover, similar metadata could be added to store default values of a property. These default values would then be assigned to a property once a new individual has been created.

Another vocabulary could be defined to store metadata about the inference process, including the most recently inferred superclasses of a certain class, the inferred types of an individual and information on whether an ontology contains inferred information in an asserted form. The motivation for such metadata is that classification is often a very time consuming process. Often it is more convenient to operate on inferred relationships than on the asserted ones without losing the information about what has been explicitly asserted by the user. If annotation properties are used to represent such data, then generic tools could simply ignore them.

---

[19] http://www.opengalen.org/
[20] http://dl.kr.org/dig/

While such metadata vocabularies do not have to become part of a future OWL specification, they may be addressed by specific W3C task forces to support the evolution of communities instead of incompatible, native formats.

3.6 Annotation Properties and OWL Full

Despite the value of annotation properties, in OWL DL, properties that are declared as annotation properties are greatly limited in so far that they can neither have range or domain constraints, nor can they be arranged in sub-property hierarchies. This type of information about a property enables tools to control the values that annotation properties can acquire. Without range constraints it is difficult to provide the user with appropriate input widgets. In a similar sense, it is often helpful to declare metaclasses so that classes can be categorized into types and different interfaces be provided for each type. Currently, using these features means that the ontology will be forced into OWL Full.

A common conception about OWL Full is that once an ontology is in OWL Full it cannot be used for reasoning. It is true that some reasoning interfaces simply reject ontologies in OWL Full without even attempting to understand why the user has selected OWL Full constructs. We argue that in cases such as rich annotation properties or metaclasses it is usually acceptable to convert OWL Full ontologies on-the-fly into OWL DL, simply by ignoring certain triples.

4 Conclusions and Suggestions for Future Work

Overall the acceptance of OWL in the Protégé community has been positive. Many of the common problems can be resolved by means of better education and tool support as well as by encouraging best practices. Based on our experiences, we suggest a number of extensions to a future version of OWL:

- Integration of user-defined datatypes (esp. for numeric ranges)
- Qualified Cardinality Restrictions
- Management of disjointness (`owl:AllDisjoint`)
- More flexible annotation properties (at least as best practices)
- Means of expressing possibility and optionality

Given that one of the major benefits of OWL is that it is an official standard, it would be counter productive to the Semantic Web if tools are “forced” into defining proprietary extensions.

Acknowledgements. This work was supported in part from a contract from the U.S. National Cancer Institute and by grant LM 007885 from the U.S. National Library of Medicine. Additional support was granted by the Semantic Mining Network of Excellence (NoE 507505) sponsored by the European Commission, and by the CO-ODE/HyOntUse (GR/S44686/1) projects sponsored jointly by the UK Joint In-

frastructure Services Committee (JISC) and UK Engineering and Physical Sciences Research Council (EPSRC).

References

1. Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The Protégé-OWL Plugin: An Open Development Environment for Semantic Web Applications. Third International Semantic Web Conference, Hiroshima, Japan (2004)
2. Noy, N., Fergerson, R., Musen, M.: The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. 2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW), Juan-les-Pins, France (2000)
3. Pan, J., Horrocks, I.: OWL-Eu: Adding Customised Datatypes into OWL. In Proc. of the Second European Semantic Web Conference (ESWC) (2005)
4. Rector, A.: Defaults, context and knowledge: Alternatives for OWL-Indexed Knowledge bases. Pacific Symposium on Biocomputing (PSB), Kona, Hawaii (2004)
5. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang , H., and Wroe, C.: OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Whittlebury Hall, Northamptonshire, UK (2004)
6. Rector, A.: Normalisation of ontology implementations: Towards modularity, re-use, and maintainability; Proceedings Workshop on Ontologies for Multiagent Systems (OMAS) in conjunction with European Knowledge Acquisition Workshop, Siguenza, Spain (2002)
7. Wolstencroft, K., Brass, A., Horrocks, I., Lord, P., Sattler, U., Stevens, R., Turi, D.: A Little Semantic Web Goes a Long Way in Biology. 4th International Semantic Web Conference (ISWC), Galway, Ireland (2005)