Architecture for an Ontology and Web Service Modelling Studio

Holger Lausen and Michael Felderer

DERI Digital Enterprise Research Institute
{holger.lausen,michael.felderer}@deri.org

Abstract. This paper outlines the architecture of a WSMO Studio, that aims at developing formal specification according to the WSMO Meta Model. Developing formal specification according to a specific formalism is not an easy task. As soon as the descriptions get numerous, large and different authors are involved manual editing without tool support does not scale anymore. In the field of ontology editors are already various tools available, however derivations in the underlying formalism and the special extension needs for a WSMO Studio require a custom development. In this paper we outline the concrete functional requirements and propose an open architecture for a studio based on the analyzes of existing work.

1 Introduction

The Web Service Modelling Ontology (WSMO) [18] defines the conceptual elements that have to be expressed in order to describe and use Semantic Web Services. Those are Ontologies, Mediators, Web Services and Goals. A component based approach can enable integration and reuse of common functionality, so that this work can be taken as the basis of future extension, e.g. for a versioning system or for the integration of different reasoners for various tasks.

The aim of this paper is to outline the exact functional requirements as well as the architecture we are using to accomplish these requirements. Thereby we will take into account existing work mainly in the area of ontology management and finally outline an open architecture for a WSMO Studio.

2 Related Work

Several editors for ontologies have been already developed [9], especially Protégé2000 [8] has gained a considerable user community. Its success has been build up on 1) its open source model, 2) its extensible plug-in architecture and 3) its flexible Meta Model. Thus it was a natural starting point for our first implementation [14]. It turned out that it was easy to capture the core of the ontological model of WSMO and to reuse Protégé features that allow the basic modification of ontologies. Based on previous work [20] it was also possible to provide a basic connection to an inference engine. Additionally we integrated a basic editor for logical expressions, with syntax completion and validation. However besides the positive effects of reuse also principle problems have

been encountered: The OKBC [5] compatible Meta Model is able to capture concepts, attributes of concepts and instances, however n-ary relations, functions, variables and especially their axiomatizations, as well as the concept of mediators turned out to be difficult to capture. More over we have encountered software engineering related problems, when designing the user interface. The event model and some of the user interface elements have been difficult to adopt. Other Web Service specific aspects like the interface description we did not attempt to implement, but foresee problems, since these elements have only very little in common with Protégé general purpose of ontology editing and knowledge acquisition.

A second WSMO Editor has been developed in the course of the SWWS project [7]. This custom Java/Swing application is developed based on the epistemological foundations of WSMO and provides graphical means to modify each modelling element. In addition the SWWS Studio provides an editor for orchestration. However its design as proprietary application does not allow extensions, e.g. if a special syntax export or some specific restrictions on a modelling elements are needed, no modifications can be made to this tool by a 3rd party.

Both approaches have not been able to fully meet the expected functionality. In order to develop an architecture able to meet the requirements we will elaborate the detailed requirements in the next section, always keeping in mind related work in the area and the approaches taken there.

3 Functional Requirements

As outlined in the introduction there are already many existing ontology editors and storage environments, however we believe that the extension and modification needs of them exceed the benefits of reuse. In this section we will outline the specific requirements on the Editor.

The main functionality from an Editor is to provide the means to create, modify and delete every modelling element of the domain of discourse according to a specific formal meta-model as well as store and retrieve those descriptions. Hence its functionality is heavily depending on the underlying formalism. A WSMO Editor consequently needs to handle Ontologies, Goals, Mediators and Web Services. For Goals, Mediators and Web Service capabilities it is required to edit complex logical formulas, whereas for ontologies it should be in principle possible to reuse existing work in the area of ontology editors.

We are first outlining for each element of WSMO the specific needs raising from the Meta Model and then discuss more general requirements of such an environment.

3.1 Ontologies

Most implementation work in this area is motivated by having a particular meta-model or development methodology in mind. Recent efforts are mainly based on the OWL respectively Description Logic primitives, as such have become increasingly important in the course of its standardization [2]. The resulting editors, such as SWOOP [12] or the

DL-Workbench [13] use a very specialized meta model, able to express concepts, properties and individuals, however the ability to edit axioms is limited by the underlying description logic expressions and thus very specialized and not as general as WSMO requires. More general approaches that have been e.g. taken by Protégé are based on OKBC [5] and by intention have very limited support for axioms. [16]. We now briefly discuss all modelling elements for ontologies and the requirements how they should be handled.

Global Issues WSMO is based on the principle of URIs (Uniform Resource Identifiers). Furthermore it supports Literals and basic XML Schema Data Types. In addition WSMO foresees an extensible set of non functional properties for every modelling element. This is different to existing approaches such as OWL [2] where only the top level component (Ontology) has a more extensive set of meta data and the sub elements such as concepts and properties only have a limited set (e.g. label or comment). One could argue that the set within other languages is extensible, however the current tool support does not encourage this and this that feature is not used.

Concepts This element can be handled very similar to how it is done in existing approaches, however a small difference is the treatment of attributes. Generally they are treated as first class citizens of the ontology and not of an concept (although it can be emulated by range restrictions).

Relations Relations are not restricted in its arity like for example in OWL which only allows binary relations. The WSMO Studio needs to be able to define relation signatures with named and typed parameters as well as their axiomatization (see section axioms below), whereas current tools only support binary relations with primitive range and domain restrictions.

Functions As relations also functions have named and typed parameters and in addition a range restriction which defines its return value. Like the general notion of n-ary relation and its axiomatization is not present in most of the discussed existing work on editors.

Instances The general notion of instances is present in most existing tools, however it refers usually only to the notion of instances defined in the syntactical framework, their acquisition and verification. In WSMO this notion is kept much broader and it foresees the notion of an external instance store and enables the connection to e.g. a relational database. Thus the studio needs to provide a way to define instances within the given framework, but also to enable a plug-in architecture that allows the connection to external sources.

Axiom Axioms are one of the most challenging parts for this editing environment. Generally a User Interface should hide the complexity of the underlying syntax, however in the case of axioms and their editing it is difficult to simplify them by using

concepts like drop down boxes and wizards and to remain the flexibility to define full complex logical-expressions according to the specific formalism. This also shows the comparison with existing tools, e.g. the frame based meta model of Protègè allows only the definition of a restricted number of logical assertions (e.g cardinality restrictions are possible but quantifications are not possible). The current work in WSMO foresees different formalisms for logical-expression, it can be full first order logic or limited to specific Description Logic or Horn Logic subset. Thus the Studio must have the ability to edit axioms and to allow different extensions of this editing feature according to the expressivity of a certain language subset. It should support syntax highlighting, validation and completion and it may support UI abstractions form the underlying formalism by Wizards or the like.

3.2 Mediators

Mediators are a novel feature of WSMO, differentiating it form most other approaches that ignore heterogeneity for the sake of simplicity, but therefore ignoring real world settings.

OO-Mediators Ontology Mediators can be used to overcome the mismatches in different ontologies and increase the reuse of existing ontologies. A WSMO Studio must be able to define Mediators according to the already specified elements. It must be able to handle the simplest type of an OO-Mediator which enables the complete import of another ontology without the need of mediation, such that for example during the creation of an logical expression for a goal, the terminology specified by the import statement is available. The studio should be able to offer extension points for (semi-) automatic merging or alignment. It should also provide means to dynamically execute more complex mediators, such as the syntax translation or basic mapping rules, such that the mediated ontologies become available to the user of the WSMO Studio.

WW, GG, WG-Mediators These Mediator have been currently defined by source and target component and the mediation service which performs the actual Mediation task. Currently these Mediators have not been fully specified and not been illustrated in an use case, they are one of the less well defined elements of WSMO. The WSMO Studio must expect some changes in their definition and usage and allow to plug-in different components.

3.3 Goals

The main elements of goals are post-conditions and effects. These are defined by logical expressions. Currently those expressions are not restricted, but it can be foreseen that their might be different levels of expressivity, as in general already mentioned for axioms (see section 3.1). Furthermore structural restrictions might be enforced in certain dialects of WSML, for example that a goal post-condition has to be fact or the capability has to be a rule. Thus the Studio must provide a basic way to allow unrestricted use of logical-expressions within the goal, it must provide extension points to allow customized editing of that properties.

3.4 Web Services

In WSMO the web service functionality is described by means of its capability; for the fields pre-conditions, assumptions, post-conditions logical expression have to be edited. Here the same requirements then outlined in the previous sections on Goals hold.

Interfaces An interface describes how the functionality of the service can be accessed by providing a twofold view on the operational competence of the service: The choreography that decomposes a capability in terms of interaction with the service (service user's view) and the orchestration decomposes a capability in terms of functionality required from other services (other service providers' view). Both elements are currently not fully specified, however we can make some basic assumptions on the requirements by considering the current technologies in those areas.

The interface enables a requester to invoke the service, generally the formalism used for its description are called Interface Definition Languages (IDL). In the area of Web Services the relevant standard is the Web Service Description Language (WSDL) [6]. However an interface contains further aspects besides the one contained in WSDL such as information about the behavior (and interrelation) of operations, which can be specified by e.g. an abstract state machine (ASM) [3]. We can identify as requirement that a WSMO Studio must be able to handle Interface Descriptions in WSDL, it must be able to extract the relevant information and present them to the user for further annotation, i.e. to map the data types to the terminology defined in the ontology and to add some behaviorally description.

Another part of the interface is the orchestration (also called private process). It defines how external services are called when performing a specific task. At the moment the concrete description elements are not defined in WSMO. Thus any workflow language could be used as proprietary solution to the problem.

3.5 Management

This section describes the requirement on the management of all the descriptions. This is orthogonal to the already mentioned top level concepts. Every Modelling or Development Environment has to deal with them.

Compatibility with existing Formalisms has to be provided if the tool shall be adopted and to enable the reuse of ontologies conforming to recent web standards like RDF or OWL. For this purpose importers and exporters are needed. Since the WSMO model is a proper superset of those standards during the import information can be preserved, whereas with the export to some of the language information may be lost. Besides for ontologies compatibility is also important for other areas like orchestration, where a translation to existing formalisms can be used to immediately use a description in a use case environment.

Inference and Validation With respect to the integration of reasoners into editing environments, there exist some interfaces, which allow basic communication between reasoners and an editing environment, the adapter layer needs to be open to enable their integration. More over the drawn inferences have to be displayed with the graphical user interface and translated to the more abstract representation form of editor, e.g. if an inconsistency can be inferred it should be marked within the tree view of the ontology which concept definition causes the inconsistency. Besides syntactical validation these inferences can be used for semantical validation.

Collaboration and Versioning is another aspect. In order to be able to work in a team on a description they need to be accessible in a shared repository, furthermore versioning support is needed to retrieve different revisions of a description and trace changes of different authors. In Software Development this is solved for example by tools like CVS [1]. It provides a centralized access to all sources, allows the retrieval of all revisions and concurrent development. Due to the lack of a special tailored system for ontologies, CVS is currently also used in ontology projects, e.g. WonderWeb [2]. We foresee the use of some management layer enabling collaboration and versioning, but will not focus on this in the beginning.

Storage and Retrieval The WSMO Studio has to support an extensible Storage and Retrieval functionality. Files in different WSML syntaxes need to be loaded and saved, as well as the possibility to connect to database back-end storage systems.

3.6 General Software Architecture Requirements

Usability. The Interface should be usable for experts as well as for novice users. That means employing a similar strategies like for example HTML Editors, that provide a WYSIWYG interface, but concurrently allow the modification of the source code as well. In section 4 and 5 we give more details on how this kind of usability can be achieved, although a full usability optimization is beyond the scope and requires external experts.

Extensibility A purely monolithic approach is not feasible, because of the complexity of WSMO and because of the different demands of the potential users. A component model is needed that allows extensions by a third party, e.g. an connector/monitor tool for an execution environment or an plug-in modifying the logical expression field editing. A WSMO Studio can for example be extended to a generic ontology tool and be reused within other projects.

4 Architecture

After we have outlined the concrete requirements we will now propose an Architecture that fits all mentioned requirements.

[1] `https://www.cvshome.org/`
[2] `http://wonderweb.semanticweb.org/`

In the field of Ontology Engineering Environments one can distinguish the architectures into two broad categories, one where the editing is build around a centralized repository and its core API (client-server) and a second where the editing is based on a stand-alone client, that provides its own API for different plug-ins [15]. E.g. WebOde [1], KAON [4], Ontolingua [10] have a client-server model. The integration of new modules is based around the API of the server and the reasoning support is directly handled by the repository. Protégé [8], OntoEdit [21] and others take the plug-in approach without requiring a centralized server (although it might be integrated by a back end plug-in). Their extensibility is based around the internal API wrapping their data model.

We believe that the second approach is the best choice for the current stage of the WSMO development since it does not require the development of a server infrastructure and allows to concentrate on the implementation of the syntax support for WSML and importers / exporters. While at the same time allowing a later extension with a more sophisticated back end (e.g. a ontology management server enabling collaborative ontology development).

Based on our experience and the analyzes of success criteria for ontology engineering environments we believe a component model for the WSMO Studio is a crucial requirement to allow extensions by different stake holders and adaption to specific needs. Furthermore a studio environment needs to provide basic support and abstraction from the underlying meta model via an API, as well as interoperability with existing standards. Based on that the architecture depicted in Figure 1 is envisioned.

The central element of our architecture is an API providing standardized access to all modelling elements of WSMO as outlined in section 3. The API will provide access to all components defined in [19].

The API will provide an adapter layer for different implementations of import / export plug-ins. This layer can convert the internal model to different representation forms, for example to the WSML user language or to an F-logic dialect that can be interpreted by an existing reasoners.

Besides plug-in for the storage also an extensible component model is needed for the graphical user interface. For this purpose we use the Eclipse platform (see section 5) and develop a WSMO runtime component for it which integrates the operations of the WSMO API into the platform, e.g. provides methods to retrieve all loaded ontologies and bind these to concepts of the component model so it can be accessed. The WSMO runtime defines the extension points where different plug-ins can be fitted that use the functionality provided by the already described core components.

By separating the API from the component platform we enable reuse of it also in applications that do not need or want to use a graphical component platform, for example a web based validation servlet.

5 Implementation

To implement the proposed architecture existing programming and ontology tools are available. Because most of these tools are Java based, Java is an appropriate implementation language for a WSMO Studio. For WSMO ontologies it should at least provide

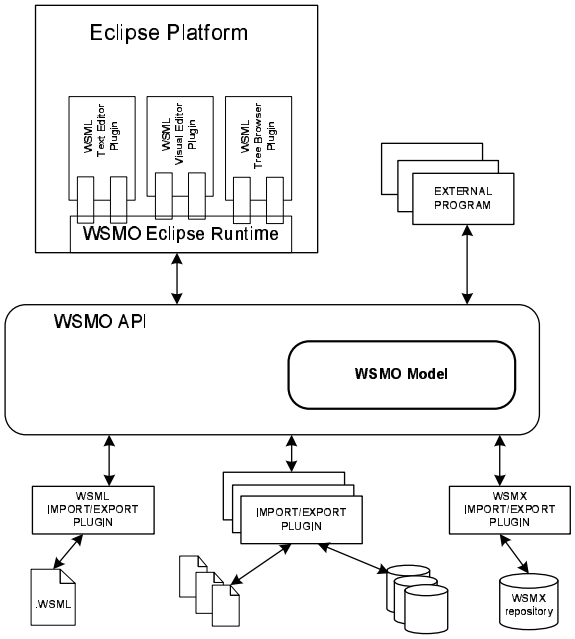

Fig. 1. Architecture of WSMO Studio

a text editor, a tree browser, a basic visual browser and extension points for integrating other plug-ins especially for logical-expressions or ontologies. The Eclipse platform [11] supports the development of a plug-in architecture with custom extension points and provides libraries to implement powerful text editors and browsers. Therefore the Eclipse Platform is an ideal platform for developing an extendable WSMO Studio. The main components of such a WSMO Studio have been shown in Fig. 1.

5.1 Eclipse based WSMO Editor

The Eclipse Platform is especially designed for building integrated development environments. The Platform's principal role is to provide tool providers with mechanisms and rules that lead to seamlessly-integrated tools. There are already some ontology tools available for Eclipse such as the Semantic Web Development Environment [3] or the 'Protègè within Eclipse' plug-in [4] which demonstrate how ontology editors can be implemented based on Eclipse. An Eclipse based WSMO Studio for editing WSMO descriptions needs at least a runtime plug-in which defines extension points for Eclipse WSMO Plug-ins and special WSMO Plug-ins.

The main functionality of the WSMO Eclipse Runtime is to define extension points for WSMO Eclipse Plug-ins and to link between the WSMO API and the Eclipse platform features. Eclipse also enables the definition of extension points for plug-ins to

[3] `http://owl-eclipse.projects.semwebcentral.org/`
[4] `http://informatics.mayo.edu/`

allow inter-module (resp. plug-in) communication. Although we implement a runtime module, other application should also have the possibility to directly plug onto the API not using the runtime.

One of the most important plug-ins for all development environments and therefore also for a WSMO Studio is a text editor for the different variants and syntaxes of WSML [17]. By using the facilities provided on the Eclipse platform a customized WSML text editor with additional features such as syntax highlighting, context and pulldown menus, key binding contexts, text outlining pages that show the hierarchical structure of the text and context behaviour can be implemented only by extending and adapting existing classes of the Eclipse library. A prototype of such an Eclipse based text editor with syntax highlighting has already been implemented (see fig.2). If one

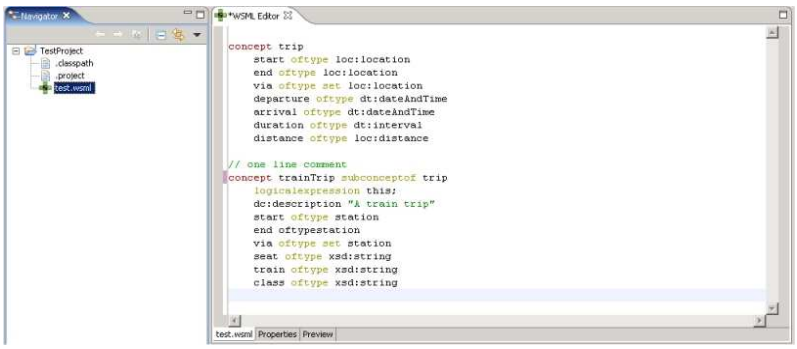

Fig. 2. WSML text editor

wants to extend the text editor with different kinds of logical expressions, different grammars for logical expressions like FOL or F-Logic can be plugged in. Other useful plug-ins for editing WSML are a tree browser and a visual editor.

By using a tree browser one can get a well structured overview and navigate through a WSMO description. In Eclipse tree browsers can easily be implemented by extending existing tree view classes. A visual editor is a optional plug-in that supports the visual modelling of WSMO ontologies. Extensions of a basic visual editor may also include objects for web services, goals and mediators. One should have the possibility to edit all properties of WSMO descriptions in the visual editor. The editor is especially for those users who don't want to code WSMO descriptions in a text editor. Besides SWT and the JFace library also the Eclipse workbench itself provides many features for implementing such a editor.

5.2 WSMO API and adapters

The WSMO API is the core part of a WSMO Studio and also reusable for other WSMO based implementations. The API abstracts from the representation of WSMO data

model and provides convenience methods. Within the SWWS Studio[5] such an API has already been developed, which will be reused and extended.

A WSMO Studio needs different import and export plug-ins to serialize WSMO ontologies into different formats. Because of the existence of many different ontology formats, it makes sense to implement an abstract adaption layer to build importers and exporters for special formats based on that layer. WSMO needs at least an import and an export plug-in for WSML - the standard representation of WSMO. Here work has already been done, by implementing a grammar and a SableCC based tree walker. Parts of that work are also used in the online WSML validator [6].

6 Conclusions and future work

In this paper we have outlined the requirements on a WSMO Studio and derived an software architecture for it while continuously comparing our approach to existing solutions. Despite our aim of maximum reuse of existing work we had to conclude that a construction up on existing ontology editors does not pay off, mainly because of the different meta model and the missing extensions points for web service related features such as orchestration development. On the other hand a monolithic and proprietary solution will not be adopted by a wide range of user, because of the lack of extension and modification possibilities. Our presented architecture takes this into account and proposes an open plug-in model based on a generic component platform such as eclipse. The first implementation steps have been taken in the form of prototyping with the eclipse platform and the development of parsers and converters for WSML and flora2. This work will be jointly continued with the developers of the SWWS Studio.

The WSMO Studio will provide a easy extension point for all researchers aiming at the realization of their research around semantically enabled web services using WSMO.

Acknowledgments This research is partially funded by the by the European Commission under the IST projects SWWS and DIP. We like to thank the WSMO, WSMX and WSML working group for fruitful input and also special thanks to OntoText for their support and discussions on the design of the studio.

References

1. J. C. Arpírez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez. Webode: a scalable workbench for ontological engineering. In *Proceedings of the International Conference on Knowledge Capture*, pages 6–13. ACM Press, 2001.
2. S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web ontology language reference. W3c recommendation, W3C, 2004.
3. E. Börger and R. F. Stärk. *Abstract State Machines—A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[5] `http://swws.ontotext.com`
[6] `http://138.232.65.151:8080/wsml/`

4. E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, N. Stojanovic, R. Studer, G. Stumme, Y. Sure, J. Tane, R. Volz, and V. Zacharias. KAON - towards a large scale Semantic Web. In K. Bauknecht, A. M. Tjoa, and G. Quirchmayr, editors, *E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings*. Springer, 2002. None.
5. V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice. Okbc: A programmatic foundation for knowledge base interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 600–607, Madison, Wisconsin, USA, 1998. MIT Press.
6. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. WSDL: Web services definition language. W3C Technical Reports on WSDL, published online at http://www.w3.org/TR/wsdl/, 2004.
7. M. Dimitrov, Z. Marinova, and P. Radkov. *Prototype Tools II (EU Project IST-2002-37134 deliverable D5.2)*. Ontotext Lab. / SIRMA, http://swws.semanticweb.org/public_doc/D5.2.pdf, 2004.
8. J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of protege: An environment for knowledge-based systems development. Technical Report SMI-2002-0943, Stanford Medical Informatics, 2002.
9. A. Gómez-Pérez et al. A survey on ontology tools. Deliverable 1.3, OntoWeb project (http://www.ontoweb.org/), 2002.
10. T. R. Gruber. Ontolingua: A mechanism to support portable ontologies, 1992.
11. O. T. International. Eclipse platform: Technical overview. Technical report, Object Technology International.
12. A. Kalyanpur. *SWOOP (Semantic Web Ontology Overview and Perusal)*. MindSwap, Maryland Information and Network Dynamics Lab Semantic Web Agents Project, http://www.mindswap.org/people/, v2.1 edition, 2004.
13. M. Kazakov. *DL-workbench - A meta-model based platform for ontology manipulation*. Open CASCADE S.A, http://projects.opencascade.org/dl-workbench/, v0.9 edition, 2004.
14. H. Lausen and M. Felderer. D9v0.1 WSMO Editor. http://www.wsmo.org/2004/d9/v0.1/, 2004.
15. R. Mizoguchi. Ontology engineering environments. Technical report, The Institute of Scientific and Industrial Research, Osaka University, 2002.
16. N. F. Noy, R. W. Fergerson, and M. A. Musen. The knowledge model of protégé-2000: Combining interoperability and flexibility. In *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, pages 17–32. Springer-Verlag, 2000.
17. E. Oren. D16.0v0.2. languages for wsmo. http://www.wsmo.org/2004/d16/v0.2/, 2004.
18. D. Roman and H. L. (Eds). D2v1.0. web service modeling ontology (wsmo). http://www.wsmo.org/2004/d2/v1.0/, 2004.
19. D. Roman, L. Vasiliu, C. Bussler, and M. Stollberg. Choreography in wsmo. http://www.wsmo.org/2004/d14/v0.1/, Apr. 2004. WSMO working draft.
20. M. Sintek. The flora query tab. http://www.dfki.uni-kl.de/ sintek/FloraTab/, 2001.
21. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. STuder, and D. Wenke. Ontoedit: Collaborative ontology development for the semantic web. In *International Semantic Web Conference - ISWC 2002*, volume 2342, pages 221–235. Springer Verlag, 2002.