Collaborative Ontology Building with Wiki@nt
- A Multi-agent Based Ontology Building Environment

Jie Bao and Vasant G Honavar

Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University, Ames, IA 50011-1040, USA
{baojie,honavar}@cs.iastate.edu

Abstract. Collaborative ontology building requires both knowledge integration and knowledge reconciliation. Wiki@nt is an ontology building environment that supports collaborative ontology development. Wiki@nt is based on a language extension to $\mathcal{SHOQ}(\mathbf{D})$ with $\mathcal{O}$ (partial order on axioms) and $\mathcal{P}$ (localized axioms in package) constructors. Wiki@nt supports integration and reconciliation of multiple independently developed, semantically heterogeneous, and very likely inconsistent ontology modules. A web browser based editor interface is provided, with features to support team work, version control, page locking, and navigation.

1 Introduction

1.1 Ontology Editing is a Knowledge Integration Process

Semantic Web aims to support seamless and flexible access, use of semantically heterogeneous, networked data, knowledge, and services. The success of the semantic web enterprise relies on the availability of a large collection of domain specific ontologies and mappings between ontologies to allow integration of data [12]. However, by its very nature, ontology construction is a collaborative process which involves direct cooperation among domain experts, knowledge engineers or/and software agents, or indirect cooperation through reuse or adaptation of previously published, autonomously developed ontologies.

In such settings, typically, different participants have only partial knowledge of the domain, and hence can contribute only partial ontologies of the domain.Common tasks involve refinement of a predefined ontology, and integration of several such partial ontologies to obtain a coherent ontology. Semantic mismatches and logical inconsistencies between independently developed ontologies are unavoidable. Thus, there is an urgent need for principled approaches and flexible tools for allowing individuals to collaboratively build, refine, and integrate existing ontologies as needed in specific contexts or for specific applications.

1.2 Proposed Approach

While there has been a great deal of work on ontology languages, inference mechanisms, as well as ontology editing environments, relatively little attention has been paid to the development of principled approaches and tools for collaborative ontology building. Existing ontology editing and discovery tools are mostly focused on stand-alone ontology development rather than collaborative construction of ontologies. In this paper, we propose Wiki@nt, a general architecture of an ontology editing, ontology refinement, and ontology integration environment. Wiki@nt exploits $\mathcal{OSHOQP}(\mathbf{D})$, a modular ontology representation language

with preference partial order on axioms; a light-weight, browser-based ontology editor which requires minimal user effort and allows concurrent editing and integration of ontologies, is presented.

2 Collaborative Ontology Building as Knowledge Integration and Reconciliation

We start with a brief discussion of the theoretical basis of Wiki@nt including logical foundations of ontology languages. We then introduce a modular representation of ontologies and discuss some problems in inconsistency reconciliation with modular ontologies.

2.1 Description Logic as a Knowledge Representation Language

Ontologies are typically described using ontology languages, such as DAML+OIL or OWL. Description logic(DL) can be used to express the formal semantics of an ontology written in those languages. A description logic consists of a Tbox and an Abox, where the Tbox is a finite set of terminological axioms such as $C \sqsubseteq D$, and the Abox is a finite set of assertional statements such as $C(a)$ or $R(a, b)$. In particular, $\mathcal{SHIOQ}(\mathbf{D})$ is the formal background DL for OWL. A complete list of $\mathcal{SHIOQ}(\mathbf{D})$ and OWL/DL constructors can be found in [10].

However, ontology languages with $\mathcal{I}$ (i.e. inverse roles) constructor suffers from complexity and/or intractability problems when combined with $\mathcal{O}$ or $(\mathbf{D})$. Hence, we use a subset of $\mathcal{SHIOQ}(\mathbf{D})$, $\mathcal{SHOQ}(\mathbf{D})$, as the basis for a collaborative ontology development environment.

We assume that we have an abstract domain $\triangle^{\mathcal{I}}$, and a set of data types D and associate with each $d \in D$, a set $d^D \subseteq \triangle_D$ where $\triangle_D$ is the domain of all types. An example Animal Ontology is given here:

```
SubClassOf( Dog , Carnivore )
SubClassOf( Dog , Pet )
SubClassOf( Carnivore, Animal)
  restriction(eats allValueFrom(Animal))
ObjectProperty( eats) domain( Animal )
individual ( billy type(Dog))
```

2.2 Package-Extended Ontology

Collaborative ontology building demands modularized ontology representation by its very nature. Current ontology languages like OWL, while offer some degree of modularization using XML namespaces, fail to fully support modularized semantics. In our previous work [3], we have argued for package based ontology language extensions to overcome these limitations. In the resulting ontology language $\mathcal{SHOQP}(\mathbf{D})$, a **package** is an ontology module with clearly defined access interface. Mapping between packages is performed by **views** which define a set of queries on the referred packages. Semantics are localized by hiding semantic details of a package with appropriately defined **interfaces** (special views). Packages provide an attractive way to compromise between knowledge sharing and knowledge hiding in collaborative design and use of ontologies.

Table 1 gives the syntax and semantics of $\mathcal{P}$ constructors. Let P be the set of all packages. We define $\triangle_P$ as the domain of P. We assume that the domain of

interpretation of all packages $\triangle_P$ is disjoint from the concrete datatype domain $\triangle_D$, the abstract concept domain $\triangle^{\mathcal{I}}$, the abstract role domain $\triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ and concrete role domain $\triangle^{\mathcal{I}} \times \triangle_D$. The resulting *package-extended* description logic language is called $\mathcal{SHOQP}(\mathbf{D})$ where $\mathcal{P}$ stands for "package-extended".

Table 1. Syntax and semantics of $\mathcal{P}$ Constructors

Constructor	Syntax	Semantics
Package	p	$p^P \in \triangle_P$
View	v	$v^{\mathcal{I}} \in \triangle_P$
Global Pkg	p_0	$p_0 \in \triangle_P$
InPackage	R_P	$R_P^{\mathcal{I}} \subseteq \triangle_T^{\mathcal{I}} \times \triangle_P$
HomePackage	$HP(t)$	$HP(t)^{\mathcal{I}} = \{p \mid (t^{\mathcal{I}}, p) \in R_P^{\mathcal{I}}\}$
NestedIn	$\in_N$	$\in_N^{\mathcal{I}} \in \triangle_P \times \triangle_P$, $\in_N^{\mathcal{I}} = (\in_N^{\mathcal{I}})^+$
SLM	SLM(t,p)	$p \in \triangle_P$ can access $t \in \triangle_T^{\mathcal{I}}$iff SLM(t,p)=true
	public(t,p)	$\forall p$, public$(t,p) = true$
	private(t,p)	$\forall p$, private$(t,p) := (p = HP(t))$
	protected(t,p)	$\forall p$, protected$(t,p) := (p = HP(t)$ *or* $p \in_N HP(t))$
Import	$im(P_1, P_2)$	P_2 is imported into P_1

2.3 Ontology Reconciliation

As noted earlier, semantic mismatches and possible logical inconsistencies between independently developed ontology modules make the combining of such modules a challenging task. Specifically, in the case of two ontology modules α, β, it is possible that although $\alpha \vDash t$, the module resulting from combining α and β may not entail t i.e., $\{\alpha, \beta\} \nvDash t$ That is, any system for collaborative ontology building has to provide mechanisms for handling *nonmonotonicity*.

An example (adapted from [9]) illustrates this problem. A dog is carnivore; however, a sick dog sometimes eats grass. Formally, we add new axioms to the Animal Ontology:

```
DisjointClasses(Plant,Animal)
SubClassOf( SickDog, Dog)
   restriction(eats someValueFrom(Plant))
```

The resulting knowledge base will be inconsistent because a sick dog (which is a dog) now can eat grass (which contradicts the assertion that dogs are carnivores). Several techniques have been developed to reconcile inconsistent ontology system, such as default logic [2] and defeasible logic [13] [9]. Here we extend our $\mathcal{SHOQP}(\mathbf{D})$ with the $\mathcal{OSHOQ}(\mathbf{D})$ [13]. An axiom is said to be *defeasible* if some other axiom could *defeat* (or override) it. The resulting ontology language is called $\mathcal{OSHOQP}(\mathbf{D})$ where $\mathcal{O}$ denotes a strict partial order on the axioms.

Definition 1 *A $\mathcal{OSHOQP}(\boldsymbol{D})$-knowledge base is a tuple $\langle \mathcal{T}, < \rangle$, where $\mathcal{T}$ is a $\mathcal{SHOQP}(\boldsymbol{D})$-knowledge base and $<$ is a strict partial order between axioms of $\mathcal{T}$. For each pair $a_1 < a_2$, a_2 is said to be **defeasible** while a_1 is a (possible) **defeater** of a_2*

For example, if we revisit the Animal Ontology in $\mathcal{OSHOQP}(\mathbf{D})$, The terminology $\mathcal{T}$ could be rewritten as

$package(1)$	$package(2)$
(1a) $public(Dog, 1)$	(2a) $im(2, 1)$; `import package 1`
(1b) $1 : Dog \sqsubseteq 1 : Carnivore$	(2b) $public(Plant, 2)$
(1c) $1 : Dog \sqsubseteq 1 : Pet$	(2c) $2 : Plant \sqcap 1 : Animal \sqsubseteq \bot$
(1d) $public(Animal, 1)$	(2d) $2 : SickDog \sqsubseteq 1 : Dog$
(1e) $public(eats, 1)$	$\sqcap \exists 1 : eats.2 : Plant$
(1f) $1 : Carnivore \sqsubseteq 1 : Animal$	
(1g) $1 : Carnivore \sqsubseteq \forall 1 : eats.1 : Animal$	
(1h) $\{1 : billy\} \sqsubseteq 1 : Dog$	

A simple combination of packages 1 and 2 is inconsistent on (1g) and (2d). However, with a partial order $<$, this logical inconsistency can be eliminated. One such possible partial order is (2d)$<$(1g) (read as axiom (2d) is *stronger than* axiom (1g)). In this case, a specific axiom (2d) *defeats* the general rule (1g). When there is a logical conflict between a pair of axioms, the weaker of the two is discarded. More details of $\mathcal{OSHOQP}(\mathbf{D})$ and its reasoning problems could be found in the unabridged version of this paper [4]

3 Architecture

$\mathcal{OSHOQP}(\mathbf{D})$ gives us an expressive language to build ontology from autonomous, distributed, and possibly inconsistent ontology modules. Wiki@nt is the implementation of an ontology editor based on $\mathcal{OSHOQP}(\mathbf{D})$ to support collaborative ontology building by a community of autonomous domain experts, organizations, or even software agents.

The name "Wiki@nt" suggests that it has a wiki-like editing environment. Wiki is originally a collaborative documentation writing/website building tool. Typical wiki system includes a script language (usually a simplified subset of HTML tags), a set of wiki pages written in the script language and shown in translated HTML pages, a RCS version control system to record modification of contents, an user profile and concurrent conflict management system to enable multiple user editing the same contents, a content navigation system such as showing link-in and link-out pages, and a simple-to-use, browser-based editing environment to generate or modify content on the fly.

We find that those features are quite desirable in a collaborative ontology editor. While most widely-used ontology editors, such as Protege and OilEd, work very well for developing a single ontology module, they do not lend themselves to collaborative ontology building. This is due to the lack of a built-in formalism to support modular ontology representation, and the lack of support for communication and cooperation among multiple individuals in editing a shared ontology consisting of multiple, independently developed modules. To overcome those deficiencies, we propose using wiki to edit $\mathcal{OSHOQP}(\mathbf{D})$ ontology. An ontology module is composed of one or more wiki pages; multiple users can edit the same content, with version control and transaction management; ontology are loaded into or uploaded from a set of wiki pages and managed by an ontology repository. Figure 1 shows the architecture of Wiki@nt.

3.1 Wiki Engine

A wiki engine should do the tasks of

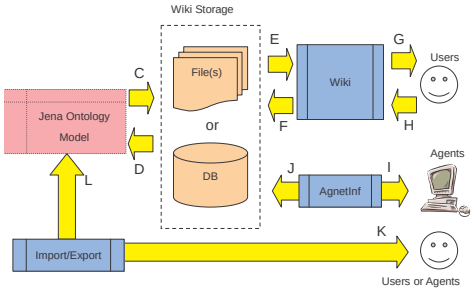

Fig. 1. The Architecture of Wiki@nt

- Provides a web interface for ontology editor and browser.
- Translates the ontology markup script to HTML for the web browser.
- Manages the storage of wiki pages, in plain file or database.
- Provides version control. When a modification for an axiom is submitted, the previous version is stored and could be restored when the committed version is found incorrect or impropriate.
- Provides transaction management.
- Generates reference report for wiki pages. Terms be used in an axiom group, and other groups that referring this group, are listed for browsing purpose.
- Generates a RSS feed for ontology repository updates.

The wiki engine we utilized is based on the JSPWiki(http://www.jspwiki.org) and implemented in Java and JSP.

3.2 Agent Management

Each participant in Wiki@nt is considered as an agent. Agent is assigned with different privileges, such as ontology administrator and package manager. Agent could join the editing of any existing ontology module or create new module.

Although our current design of Wiki@nt does not include concrete design of software agents, we do reserve an RPC interface that enables agents to communicate with Wiki@nt. The reason is while fully automatic ontology construction or mapping are still impossible, software agents can assist humans in several aspects of collaborative ontology development e.g., finding useful concepts and relations among concepts from original data sources. Small pieces of ontologies, such as consistent concept (term) in data or concurrence of two concepts, can be generated by software agents. The results may be subjected to review of domain experts, or even other software agents. Thus, in principle, it is possible for software agents to participate in collaborative ontology building using Wiki@nt.

3.3 Ontology Markup Script

We defined a set of markup script tags to correspond to the syntax of the ontologies. When a wiki page is under editing, its wiki markup script is loaded and translated to user friendly text, such as HTML web page. The syntax is a extension to OWL to support package and partial order on axioms. Wiki markup script is a human readable syntax equivalent to the N-Triple syntax. N-Triple syntax is an alternative to the RDF/XML syntax and each line in N-Triple serialization is a triple statement with subject, predicate and object.

For example, axiom SubClassOf(Dog,Carnivore) in the Animal Ontology could be represented by N-Triple syntax as: <http://mydomain.org/animal#Dog> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://mydomain.org/animal#Carnivore>, or in short form <animal:Dog> <rdfs:subClassOf> <animal:Carnivore>. It's wiki script is [animal:Dog] [rdfs:subClassOf][animal:Carnivore]

Each axiom is assigned a URI (uniform resource identifier) as label. Thus, for example, http://mydomain.org/animal/package1#Dog represents $Dog \sqsubseteq Carnivore$ in package(1), Animal Ontology.

User can create a new page or modify the source script of an existing page. The editing action is assisted by several wizards (such as class creating wizard) and a browser (eg. Show subclass and superclass of the class in question).

3.4 Memory Management

While most of the popular ontology editors have in-memory model for edited ontology, Wiki@nt doesn't maintain in-memory model for each resident ontology for several reasons: An in-memory model limits the scalability of the system with respect to both the axioms number in one ontology and the number of ontologies in the Wiki@nt ontology repository; In-memory model implicitly assumes the existence of a global ontology during the ontology development process and requires monotonic behavior of the ontology - neither of these assumptions is desirable in a collaborative ontology building scenario. In short, creating a centralized ontology model in memory defeats the very purpose of having a modular ontology

Note that that even when the size of the ontology in question is huge, usually only a small fraction of its axioms are involved during an editing action. Hence, we store the ontology as a set of separate, possibly distributed blocks in Wiki@nt. Each block is serialized to external storage when it's not being actively edited, and being loaded into the memory only if it's edited or referred. A (local or remote) partial ontology model will be dynamically loaded into local memory in a reasoning process only if it is needed. The partial model could be a package, a small part of a package, or even an axiom. This is inspired by widely used techniques of database memory management where partial content of the database is dynamically loaded and unloaded to allow manipulation of of a much larger volume of data than can fit in limited memory.

Another technique to reduce memory burden on Wiki@nt server is to build the dynamically loaded model on client side instead of server side. This is reasonable since the dynamically loaded model is just a local-interested part of whole ontology. Technically, this is done by offering a Java Applet interface to read and update Wiki@nt pages.

3.5 Modularization of Ontology

Ontology stored in Wiki@nt is managed on package level and block level. A package is a logic module for an ontology, usually from a single participant. An ontology could be composed by several packages, and one package could be reused by multiple ontologies.

A block is a set of related axioms inside a package, and will be physically mapped to a wiki page. A package will include one or more such blocks(pages).

Although different decompositions of an ontology package are logically equivalent, the size of each ontology block will affect the convenience and efficiency of ontology editing and reasoning. It should not be too big (i.e. the whole package), or too small (e.g., a single triple). In Wiki@nt, we choose axiom groups as ontology blocks. Each axiom group contains triples with same subject. For example, the axiom groups in Animal Ontology package(1) will be *Dog*, *Carnivore*, *eats*, and *billy*. Restrictions and anonymous classes, are assigned to the terms from where they are referred. Each axiom group is translated to wiki markup script and stored as a wiki page. An ontology could be stored distributedly in multiple pages, physically in file or database, and could be dynamically, partially loaded when necessary.

3.6 Ontology exporting/importing

When an ontology is needed e.g., for reasoning, we export wiki pages as a single ontology file or read an ontology file into Wiki Ontology Repository. The relevant portion of an ontology is extracted or assembled from the wiki pages. We use the Jena toolkit to create the in-memory model and as parser/writer for ontology files.

Each loaded ontology is assigned a unique name, eg. http://mydomain.org/animal/, and it's member packages, eg. http://mydomain.org/animal/package1, are registered to that ontology. It's also possible that packages from different ontologies could be reassembled into a new ontology, thus provide a flexible way for ontology reuse and integration.

3.7 Reconcile the Inconsistency

Inconsistency among modules should be resolved when integrate those modules. In Wiki@nt, we assume each package should be consistent. A partial order can be specified on package level, eg. [Package1] [wiki:stronger][Package], which means all axioms in Package1 are stronger than Package2; it can also be specified on axiom level, like [Package1:1g] [wiki:stronger][Package2:2d].

The specification of the partial order $<$ among modules and axioms may be based on principles of:

- reliability of the source of module/axiom
- the social order of the author of module/axiom
- A more recent module/axiom may be preferred over an earlier one;
- exceptions are stronger than the general rules.

Wiki@nt defined two default defeating rules if user not specifies otherwise. First, ut assigns higher priority to local package axioms relative to axioms from imported packages in cases where a local package can be seen as an extension or an exception to a general ontology. Other partial order assignment policies is based on the social order of the agents in the Wiki@nt community, such as ontology administrator, package manager and common user.

3.8 Transaction Management

Transaction management is to ensure consistency of module and protect critical resource from multiple access. It is widely used in DBMS and certain ontology

editor, such as OntoEdit [14]. However,OntoEdit only allow locking of concept hierarchy.

Wiki@nt denies the write-access of agents to a page and related pages if it is locked by some other agents. Following strategies are used for determining what pages should be locked:

- If a concept is under editing/locking, its superclasses in the class hierarchy will be locked.
- If a property is under editing/locking, its superproperties in the property hierarchy will be locked.
- If a instance is under editing/locking, its belonging class will be locked.
- If a concept, property or instance is under editing/locking, all other concepts, properties or instances in the same page(axiom group) will be locked.
- If a package is state as being locking, all importing package will be locked.

Locking could be propagated by recursively apply above-mentioned strategies.

4 Summary and Discussion

4.1 Related Work

Collaborative Ontology Editor A number of ontology editors have been reported in the literature [6,11]. However, most existing ontology editors including the most widely used ontology editors Protege and OilEd provide little support for collaborative ontology development. Representative editors that support collaborative ontology editing include CODE [8], KAON [5], OntoEdit [14], Ontolingua [7], and WebODE [1] . Most of them provide concurrent access control with transaction oriented locking, and in some cases, even rollback. However, none of the existing ontology editors, to the best of our knowledge, provides principled approaches for manipulating independently developed, semantically heterogeneous ontology modules or for reconciling logical inconsistencies between such modules.

One advantage of Wiki@nt over reported editors is its truly distributed nature for storage of ontology modules. While most editors requires in-memory model for current-editing ontology, Wiki@nt dynamically load partial model only when it is necessary, and/or share the memory burden between the server and clients.

Another distinct feature of Wiki@nt is its wiki-based design thus enable it borrow many well-stablished feature of wiki system, each as user-friendly and easy-to-use interface, version control, user management, page locking, translating from script to HTML, and persistent storage in relational database.

Collaborative Knowledge Base Construction Some collaborative knowledge base construction projects, although not focused on ontology building, address similar problems. **Nooron**(http://www.nooron.org) is a knowledge publishing system and has a wiki for ontology browsing. **MnM** [15] is an annotation tool which provides both automated and semi-automated support for annotating web pages with semantic contents. MnM integrates a web browser with an ontology editor and provides open APIs to link to ontology servers and for integrating

information extraction tools. However, it doesn't have concurrent access control. **FoaF**(http://www.foaf-project.org/) is an acronym for "Friend of a Friend" , an experimental project and vocabulary for the Semantic Web. The project is open and allows participants to add their own information. The result is an RDF based knowledge base containing contact and acquaintance information about the participants. **WikiPedia** (http://en.wikipedia.org/) is a wiki-based open-content encyclopedia that is editable by participants. Articles in WikiPedia are written in natural language, and the relation between items is not formal. Nevertheless, articles can be seen as concepts and links between them seen as properties among them, in a informal sense.**Open Directory Project** or called **DMOZ** (http://www.dmoz.org/) is an online, open, collaborative taxonomy building project for web catalog. Now it has a taxonomy tree of over 590,000 categories and over 4 million classified sites. The relations between DMOZ concepts is strict "subClassOf".

Although these projects lack formalized and full-fledged ontologies, they offer interesting demonstrations of the feasibility of collaborative ontology development. Wiki@nt proposed in this paper is inspired by the success of DMOZ and WikiPedia, and aims to provide support for such efforts using a formal ontology language to facilitate machine interpretable annotations of data.

4.2 Summary and Outlook

In this paper we have described

- The ontology representation formalism to support modularity and axiom order.
- A distributed ontology representation and storage methodology based on wiki.
- A Light-weight ontology editor to support collaborative ontology building.
- Important issues in wiki-based ontology editing, such as transaction management, memory management, agent management, modularization of ontology and inconsistency reconciliation.

Some interesting directions for future work include:

- Incorporation of more advanced transaction management and incorporation of safe mechanisms for handling simultaneous editing and modification of ontologies
- Investigation of useful policies for assigning partial order among axioms, including those that are base on machine learning or probabilistic approaches.
- Applications of collaborative ontology building environments for information integration from autonomous, distributed, semantically heterogeneous information sources.
- Detailed study of scalability of Wiki@nt, test it with big ontologies such as WordNet.

Acknowledgments

This research is supported in part by grants from the NSF (0219699) and the NIH(GM 066387) to Vasant Honavar

References

1. J. C. Arpirez, O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez. Webode: a scalable workbench for ontological engineering. In *Proceedings of the international conference on Knowledge capture*, pages 6–13. ACM Press, 2001.
2. F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. Technical Report RR-93-20, 1993.
3. J. Bao and V. Honavar. Ontology language extensions to support localized semantics, modular reasoning, and collaborative ontology design and ontology reuse. Technical report, TR00000341, Computer Sicence, Iowa State University.
4. J. Bao and V. Honavar. Collaborative ontology building with wiki@nt - a multi-agent based ontology building environment. Technical report, TR00000343, Computer Sicence, Iowa State University, 2004.
5. 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*, volume 2455 of *Lecture Notes in Computer Science*, pages 304–313. Springer, 2002.
6. M. Denny. Ontology building: A survey of editing tools. Technical report, O'Reilly XML.com, November 06, 2002.
7. A. Farquhar, R. Fikes, W. Pratt, and J. Rice. Collaborative ontology construction for information integration, 1995.
8. P. Hayes, R. Saavedra, and T. Reichherzer. A collaboration development environment for ontologies. In *Proceedings of the Semantic Integration Workshop, Sanibel Island, Florida,*, 2003.
9. S. Heymans and D. Vermeir. Using preference order in ontologies, 2002.
10. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1), 2003.
11. Ontoweb. Deliverable 1.3: A survey on ontology tools.
12. J. A. Reinoso-Castillo, A. Silvescu, D. Caragea, J. Pathak, and V. G. Honavar. Information extraction and integration from heterogeneous, distributed,autonomous information sources - a federated ontology-driven query-centric approach. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, 2003.
13. D. V. S. Heymans. A defeasible ontology language. In e. a. E. R. Meersman, Z. Tari, editor, *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE : Confederated International Conferences CoopIS, DOA, and ODBASE 2002, Lecture Notes in Computer Science*, volume 2519, pages 1033–1046. Springer-Verlag Heidelberg, 2002.
14. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. OntoEdit: Collaborative ontology development for the semantic web. In *Proceedings of the first International Semantic Web Conference 2002 (ISWC 2002), June 9-12 2002, Sardinia, Italia.* Springer, LNCS 2342, 2002.
15. M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. Mnm: Ontology-driven tool for semantic markup. In S. Handschuh, N. Collier, R. Dieng, and S. Staab, editors, *Proceedings Workshop on Semantic Authoring, Annotation and Knowledge Markup (SAAKM 2002)*, pages 43–47, 2002.