ModOnto: A tool for modularizing ontologies

Camila Bezerra, Frederico Freitas, Jérôme Euzenat, Antoine Zimmermann

ModOnto: A tool for modularizing ontologies

Camila Bezerra[1], Fred Freitas[1], Jérôme Euzenat[2], Antoine Zimmermann[2]

[1] Informatics Center, Federal University of Pernambuco
P.O. Box 7851, 50732-970, Recife – PE – Brasil
{fred, cbs}@cin.ufpe.br
[2] INRIA, Rhône-Alpes, Montbonnot, France
{Jerome.Euzenat, antoine.zimmermann}@inrialpes.fr

Abstract. During the last three years there has been growing interest and consequently active research on ontology modularization. This paper presents a concrete tool that incorporates an approach to ontology modularization that inherits some of the main principles from object-oriented software engineering, which are encapsulation and information hiding. What motivated us to track that direction is the fact that most ontology approaches to the problem focus on linking ontologies (or modules) rather than building modules that can encapsulate foreign parts of ontologies (or other modules) that can be managed more easily.

Keywords: ontology, modularization, reuse, composition.

1 Introduction

The realization of the Semantic Web depends on a number of factors, one of which is the ability to reuse ontologies [9]. Ontology construction is deemed to be a time-consuming and labour-intensive task. Therefore, it heavily relies on the possibility of reusing existing ontologies. Nonetheless, ontologies and ontology libraries, for instance, the Ontolingua server [26] and the recent OntoSelect ontology library [27], are available, together with the first search engines for the retrieval of Web ontologies, such as Swoogle [13], Ontosearch [23] and Watson [28], although even Google can be used, when users perform queries with keyword filetype:owl or filetype:rdfs. However, from an Ontological Engineering perspective it can be hard to reuse ontologies, especially when the ontologies being reused are large.

Currently, there are two major ways of reusing an ontology. Ontology editors such as Protégé [2] allow the reuse of another ontology by including it in the model that is being designed (in Protégé this happens through the inclusion of other projects). The web ontology language OWL [1] offers the possibility to import an OWL ontology by means of the <owl:imports> statement.

In both cases, the whole ontology has to be included. Unfortunately, ontology engineers might be interested only in a portion of large ontologies.

Regarding the former case, OWL provides the <owl:imports> construct for linking multiple OWL ontologies to form a larger OWL ontology. However, such a syntactic importing solution of OWL lacks support for partial reuse of ontology modules. The lack of support for selective reuse of parts of one ontology module limits the ability to reuse ontology modules because using the module in its entirety might introduce unwanted inconsistencies or impact performance, because of the presence of a large number of assertions that might be irrelevant in the context of the target application. For example, when a user is only interested in reusing concepts about Organizational Processes from the SUMO ontology [18], she has to import the whole ontology, which contains 20,000 terms.

Ontology modularization [12] could help overcome the problem of defining a fragment of an existing ontology to be reused, in order to enable ontology developers to include only those concepts and relations that are relevant for the application they are modeling an ontology for.

This paper presents a concrete tool that incorporates an approach to ontology modularization that inherits basically two of the main principles from object-oriented software engineering, which are encapsulation and information hiding. What motivated us to track that direction is the fact that most ontology approaches to the problem focus on linking ontologies (or modules) rather than building modules that can encapsulate foreign parts of ontologies (or other modules) that can be managed more easily.

The article is organized as follows: section 2 describes some of the main concepts regarding ontology modularization; section 3 presents our approach to ontology modularization; section 4 introduces ModOnto, our tool to implement such approach; section 5 is devoted to related work and section 6 finishes the article with some conclusions and enlists future work, some of them already under development.

2 Ontology Modularization

In the early 90's, knowledge engineering promoted the integration and massive reuse of knowledge bases written in different formalisms and representation languages from existing expert systems and agents. Shared ontologies were considered to tackle these tasks and the field matured over the years.

One of the ontology subfields that evolved was ontology engineering. At first, ontology development resembled more artwork than engineering, with each team adopting its own set of principles, criteria and design phases [24]. To a certain extent, ontology building methodologies remind of system analysis. They provide guidance to developers; have similar iterative phases, like specification, implementation and evaluation, among other similarities with software engineering techniques. However, although the field has worked a lot on reuse, a culture of building blocks for ontologies, such as the component culture that took over in the realm of object oriented software engineering, has not shown up. Unfortunately, the concepts of modularity and composition have not been spread as crucial issues and principles for ontology building.

Modularization can be perceived in two different ways. On the one hand, people think of modularization as the process that leads to decomposing a large ontology into smaller modules, known as ontology partitioning [9]. On the other hand, an equally viable perception is to assume that the semantic web is filled with ontology modules and that there is a requirement to assemble some of these modules to form a wider ontology. The module in this perception is something like a building block; the starting point is the set of useful modules; the target is the new ontology. Another research aspect regarding modules is targeted to define what a module is, and what it should contain at least.

It is necessary to formalize and define what an ontology module should be, particularly in terms of its requirements. Several definitions of an ontology module exist in the literature. One remarkable fact is that, while the notion of module is quite well understood and accepted in the area of Software Engineering, it is not clear at all what the characteristics of an ontology module are [11]. A possible definition for an ontology module is the following: *An ontology module is a reusable component of a larger or more complex ontology, which is self-contained but bears a definite relationship to other ontology module [19]*.

This definition implies that ontology modules can be reused by developers either as they are, or by extending them with some new concepts and properties. Alan Rector adds to that definition by specifying three requirements that should hold for ontology modules [20]:

- Loose Coupling, which means that often nothing can be assumed about different modules, they might have very little in common (like concepts or representation language), and therefore as little interaction as possible should be required between the modules.
- Self-Containment, which means that every module should be able to exist and function (this could involve performing reasoning tasks or query-answering) without any other module.
- Integrity, which means that even though modules should be self-contained they could depend on other modules so there should be ways to check for, and adapt to, changes in order to ensure correctness.

In the next section, we describe our approach to modularization, along with its rationale.

3 The OntoCompo approach

The goal of our ongoing research is to define and implement an ontology modularization approach that provides the definition and composition of modules (or ontologies). The approach was inspired by the object-oriented components market, for it is indeed the most successful reuse approach in the computer science mainstream.

Following this stance, ontologies or modules should be seen as building blocks, smaller units of knowledge (or modules), designed for composition with other modules.

We took into consideration some design requirements while conceiving the approach. One concerns module properties; the approach should offer facilities that permit and support modules to be specified, in such a way that avoids dependence from a particular ontology or module. This would enable exported modules to be replaced by others with similar contents.

The second requirement brings the main novelty of the approach: most of the modularization proposals are more ontology linking approaches rather than modular, in the sense that they don't enforce ways of limiting the modules. We borrowed the following principles from object-orientation:

Encapsulation: enforces module independence by "hiding" implementation details to importing modules. Module specifications are described as an interface which is what can be called by importing modules. So module implementation can evolve without affecting the importing module specification and modules are replaceable by others offering the same interface. In terms of ontologies, the implementation corresponds to the set of axioms used to define it.

Separate development: Since what imported modules provide is well-defined, software developers can rely on this interface and develop their own part. This would be useful as well for ontology development in which one can concentrate on the development of part of the model while the part it is relying on are still underspecified.

Reusability: Because module specifications are descriptions of a coherent and explicit set of primitives that a module is meant to provide, a module can be reused in another similar context.

Bearing this in mind, we defined a module language which comprehends some aspects related to an organized way of implementing module composition with the above principles. Particularly, once plenty of issues are related to modularization, such as reasoning and distributed semantics, we have defined the language formally, i.e., with three syntaxes and a semantics, which are described in the next subsections.

3.1 Language Syntaxes

On the syntactic side, a module language must be able to encapsulate ontology fragments, to refer to other modules and to define the interface between these modules. The module language that we are designing takes advantage of existing building blocks for expressing these:

- ontology fragments are expressed in the OWL ontology language;
- relations between modules make use of URIs (Uniform Resource Identifiers),
- interfaces between modules are expressed through alignments among the OWL entities provided by these modules. This feature is not fully implemented yet.

The entities that are exported by and imported to modules will be named entities of the ontologies, i.e., classes, relations or individuals. This module language disposes of three syntaxes:

- an XML eXtensible Markup Language, syntax, compatible with OWL, for exchanging between programs (see example in the next section),
- a more concise human readable syntax for displaying in documents and
- a graphical syntax for ensuring the compatibility between tool displays.

The module tags that compose the (second) language syntax to define modules are displayed in Table 1 below.

Feature	Description
Id	Module identifier
Uses	List of modules
Imported-entities	List of ontology entities from imported entities
alignments	List of aligments
Content	Entity definitions and axioms
Exported-entities	List of entities from either the ontology defined in content or from the imported entities in imports
comments	

Table 1. Module Language tags and descriptions.

From this syntax, every feature represents a block of information of the module.

- The Uses Feature is the list of external modules or ontologies being used to build the new one;
- Imported-Entities are the list of entities (classes, properties and instances) from other ontologies or modules that are needed to create the module.
- Content contains the new entities created using or not imported entities. New axioms could be added as well.
- The alignments feature is needed (although not available yet). When working with different modules and mappings are needed involving exported entities.
- The exported-entities are entities from the content or from the list of the imported entities that will be available when one reuses the module being defined, i.e. , the classes, properties, axioms and individuals ready for reuse.

An example of this syntax in the XML format is detailed at subsection 4.1.

3.2 Language Semantics

On the semantic side, it is necessary to define unambiguously what is the meaning of "encapsulating" is, i.e., what is hidden (and can be changed) and what is exposed (and

should always be provided) in a module. This definition leads to a semantics for the module language, i.e., a way to determine which assertions are logical consequences of the defined modules. For that purpose we have defined a formal semantics for that language, which is available at [25], using our already existent semantics for distributed systems of ontology and alignment [22]. This semantics has several interesting features in the context of ontology modules:

- It does not depend on a particular ontology semantics, but only needs to be able to determine what are the logical consequences of an ontology. This is very useful when using modules in which the ontology languages can be different and also enables connected ontologies or modules to be replaced by similar ones;
- It can be constrained in a way that allows hiding the module implementations;
- It disposes of pioneering mechanisms to encapsulate and regulate the interface of a module, thus selecting parts which can be imported/exported.

We see at least two advantages with this modularization approach:

- It can be constrained in a way that allows hiding the module implementations;
- It disposes of pioneering mechanisms to encapsulate and regulate the interface of a module, thus selecting parts which can be imported/exported.

Therefore, the main goal of the approach is to help ontology engineers by facilitating ontology reuse or partial reuse, and by making separate development easier with encapsulation and interfaces. In particular, alignments help a lot in importing and integrating heterogeneous modules, even though they could be replaced by semantically equivalent axioms in the internal ontology.

In the next section, we describe the implemented facilities to build modules using the designed module language.

4 ModOnto: a tool for modularization of ontologies

Beyond the design of the language with its syntax and semantics, components are needed for modularization to be accepted at a larger scale, thus avoiding burden over ontology developers. We identified some of these components, that should be viewed as an integrated suite, and enlist them as follows, along with some of their features.

- **Module API** – All of the development of the remaining components will be based on the implementation of a module API able to deal with module specifications as described in the syntax. The API will provide the basic data structure and accessors as well as a parser and serializer on top of the structure.
- **Module checker** – in an environment where lots of modules are available and evolving, module definitions can be prone to errors (non aligned interface due to error or evolution). So a checker should verify if all of the definitions

needed by one module are present on its imported modules or in their own imported modules, and so on.

- **Module extractor -** Although we are departing from the assumption that a module language is designed for developers to creating modular ontologies, tools to create modules from existing ontologies are suitable as well. Thus, a user of this tool should be able to make a 'select' (or 'extract' operation) over an ontology in a number ways, such as:selecting/ruling out entities (classes, roles, individuals) to be exported using a graphical interface and selecting/ruling out entities using expressions (e.g. sub-, superclasses, complements, and others) or special buttons and parameters that represent them and generating a module description.
- **Module library** – building on the Module API, the Module library is a repository for "off-the-shelf" modules that will help developers to choose its proper modules. A module description in this library should include its imported modules and a visualization tool to see the linkage graph between modules. An aspect to be observed is, as the case of Semantic Web solutions, cyclic links are accepted.
- **Module linker** – is a graphical tool to build modules using other modules. It should interact with the module library to pick up imported modules to be included, and support the specification of necessary alignment between modules, mainly the new one.
- **Reasoner interface for modules -** Modular ontologies are ontologies. So they should be usable just as yet another ontology: being able to query if an assertion is entailed by a particular module. For that purpose, it will be necessary to adapt an actual OWL reasoner or query engine such as it can answer such queries from a modular ontology.

ModOnto has been implemented as a standalone tool, providing an intuitive graphical interface for modularization that facilitates the selection of ontology entities for building modules. The Java programming language, the OWL API from Manchester, which contains an implementation for the W3C Web Ontology Language OWL, were used to the tool. From the components above, only the first three are implemented yet. Figure 1 illustrates the logical architecture of the system.

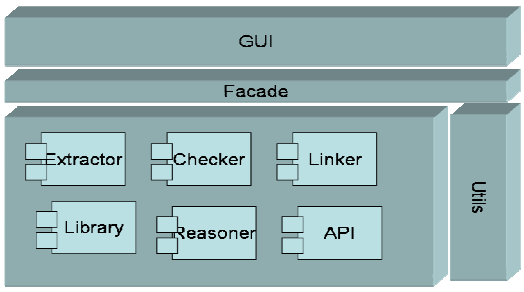

Figure 1: Architecture

This architecture has been designed to be as flexible as possible to allow for integration with already existent tools. In the figure, the GUI layer represents the

classes for graphical interfaces. The façade's aim is a twofold: (a) to leave the GUI independent from the components' implementation, and (b) to simplify the system access by abstracting many implementation details of the components' usage. The Utils layer represents the external APIs that have been used, like the OWL API and the Alignment API.

The tool's first screen contained a menu to load the desired ontology. When the user chooses the ontology, its classes and properties are shown, so that the user can select the ones she wishes to compose the new module. The tool deploys options to select/deselect sub- and super-entities (sub- and superclasses, properties and individuals) during the entities' selection. These options save time because the user can import a large set of entities and expressions at once, the process of modularization is optimized. After that, the user has the option to generate the module.

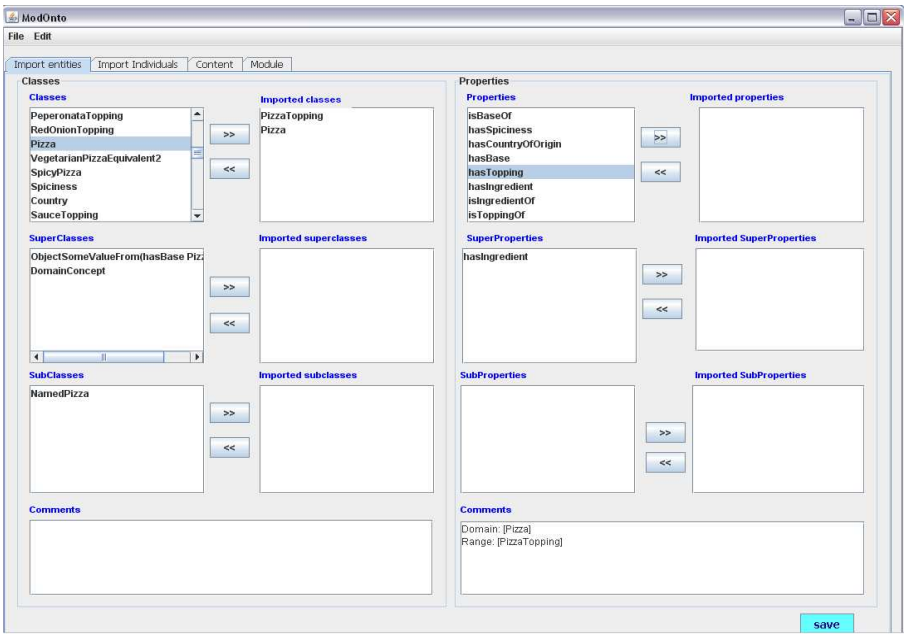

Figure 2: View of extractor component

4.1Example

We present here an example departing from the well-known ontology of pizzas [14] and ontology of countries. The first step is to select the desired entities from the ontology. We accomplish this step through the graphical interface to import the entities shown in Figure 2. Note in the figure that three panes are available, for classes, properties and instances. From them, we selected two entities, the `Pizza` class and the `hasTopping` property.

When a class is chosen to be imported, the system checks if this class was already imported even if it is a super- or subclass of this class, displaying the message "entity

already added!" if the class was already imported. After selecting the entities, the tool generates a module description illustrated in Figure 3. The generated XML module description contains the namespaces, the list of imported entities - in this case, the `Pizza` class and the `hasTopping` property - exported entities and its content.

The goal of this module is to provide a new kind of Pizza, the `NortheasternBrazilian` class, that has as Topping only `SaltedMeat`, and whose country of origin is `Brazil`. For this module the Pizza ontology was reused.

```
<?xml version="1.0"?>
<mod:MODULE xmlns="http://www.inrialpes.fr/exmo/modules#"
     xml:base="http://www.co-ode.org/modules/ModuleTest.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl11="http://www.w3.org/2006/12/owl11#"
     xmlns:owl11xml="http://www.w3.org/2006/12/owl11-xml#"
     xmlns:mod="http://www.inrialpes.fr/exmo/modules#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <uses>
        <owl:Ontology rdf:about=
"http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl"/>
        <owl:Ontology rdf:about=
"http://www.bpiresearch.com/BPMO/2004/03/03/cdl/countries.owl"/>
    </uses>
    <Imported-entities>
        <owl11xml:OWLClass rdf:about="pizza#Pizza"/>
        <owl11xml:ObjectProperty rdf:about="pizza#hasTopping"/>
         <owl11xml:OWLClass rdf:about="countries#Country"/>
        <owl11xml:OWLIndividual rdf:about="countries#Brazil"/>
    </Imported-entities>
    <Content>
       <owl:Class rdf:about="#NortheasternBrazilian">
         <rdfs:subClassOf>
             <owl:Restriction>
                <owl:allValuesFrom>
                      <owl:Class rdf:ID="SaltedMeat"/>
                </owl:allValuesFrom>
                <owl:onProperty>
                    <owl:ObjectProperty rdf:about="#hasTopping"/>
                </owl:onProperty>
             </owl:Restriction>
          </rdfs:subClassOf>
           <rdfs:subClassOf>
             <owl:Restriction>
                <owl:onProperty rdf:resource="#hasCountryOfOrigin" />
                <owl:hasValue rdf:resource="countries#Brazil" />
             </owl:Restriction>
          </rdfs:subClassOf>
        </owl:Class>
    </Content>
    <Exported-entities>
       <owl:Class rdf:about="#NortheasternBrazilian">
       <owl11xml:OWLClass rdf:about="pizza#Pizza"/>
    </Exported-entities>
</mod:MODULE>
```

Figure 3: XML description of a module generated automatically by ModOnto

5 Related Work

During the last three years, there was an increasing interest and active research about ontology modularization. In 2006, the first international event was held about this theme, the Workshop of Modular Ontologies in the International Semantic Web Conference (ISWC) with a second edition taking place in the following year. Some projects also dedicated research branches touching modularization issues. For instance, the WonderWeb initiative [21] which looked into the problem of reasoning with ontologies expressed with Description Logics and in particular in OWL, for example, ontologies expressed in the ontology language OWL. There was also a branch of the Knowledge Web Network of Excellence and NeOn project [15] especially devoted to modularization.

Moreover, there is a growing interest in OWL language constructs (or language extensions) to support modular ontologies, including several syntactic extensions to OWL including dOWL [4], C-OWL [3] and E-Connections [6], this latter being used to express relationships between different distributed ontologies in the Semantic Web [8]. However, they are limited in several ways [5]:

- Limited expressivity. C-OWL does not support linking two classes in different modules with properties, and E-Connections has no direct support for inter-module class subsumption.
- Reasoning difficulties, arising from the lack of mechanisms to prevent arbitrary domain relations in C-OWL and the requirement of strict domain disjointedness in E-Connections.

Several approaches aimed at ontology modularization with the goal of enabling reasoning about large and/or distributed ontologies in a computationally treatable and scalable way. This line of research usually consists in splitting large ontologies in small pieces, so that it is only necessary to reason with one of these parts at a time.

The problem of dividing an ontology into a number of modules has also received much attention in recent years. A number of approaches that aim at solving this problem have been proposed. The method by Stuckenschmidt and Klein[9] consists of automatically partitioning large ontologies into smaller partitions based on the structure of the class hierarchy, taking into account the internal coherence of the concepts. The method can be broken down into two tasks: the creation of a weighted graph, and the identification of partitions from the dependency graph. The weights defined in the first step determine the results of the second. The authors do not consider the semantics of the relations in their method and instead offer a “simple and scalable method based on the structure of the ontology"[9]. This is a drawback to their approach because considering what the relations mean is important in determining modules.

6 Conclusions and Future Work

Along the OntoCompo project, we have designed an ontology modularization approach inspired on one of the most successful techniques of computer science, object orientation. Our approach is primarily based on the (adapted) principles of encapsulation and information hiding. Following these principles, we developed a module language with three syntaxes and a formally well-defined semantics [22]. In this paper, we present a concrete implementation of the language, the ModOnto tool that assists users in the construction of new modules starting from ontologies or other modules. This project anticipates an ontology module market that can possibly come out as a consequence of the Semantic Web, in the same flavor as the Java components market in the Web, when module repositories become available.

We see a large road ahead as future work. In order to foster the approach and language popularity, the remaining components have to be implemented – particularly the module library (see Figure 1). The suite must be integrated with popular ontology frameworks like Protégé [17], KAON2 [16] and the ones from NeOn project [15] too.

Extending the language so as to encompass the use of expressions to select /exclude classes would also be a pro for the language and tool. We intend to look into this issue, and check the feasibility of including these constructors in the language.

Another very important aspect of the approach to be tackled soon is the creation of an ontology composition algebra that aims at building modules by applying operations to other modules, and defining the constraints to affect the axioms. In this algebra, the more traditional operations for computing union, intersection and difference between modules should be defined and provided. In particular, we should investigate the algebraic properties for composition operators, like commutatively, associatively, transitivity and replacement of ontologies in a composition.

Additional work on integration with the alignment work already defined [22] will also be accomplished so as to treat on-the-fly compositions.

7 Acknowledgements

The authors wish to thank CNPQ (National Council of Technological and Scientific Development) and INRIA (The French National Institute for Research in Computer Science and Control) for its partial sponsorship through the research project Composition and Modules for Ontology Engineering.

References

1. OWL, Web ontology language, http://www.w3.org/TR/2003/CR-owl-features-20030818.
2. N. Noy, R. W. Fergerson, and M. A. Musen, 'The knowledge model of protégé-2000: Combining interoperability and flexibility', in Proceedings of the 12th EKAW Conference, ed., R. Dieng, vol. LNAI 1937, pp. 17–32, Berlin, (2000). Springer Verlag.
3. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing ontologies. In: International Semantic Web Conference. (2003), 164-179.

4. Avery, J., Yearwood, J.: Dowl: A dynamic ontology language. In: ICWI. (2003), 985-988.
5. Bao, J., Caragea, D., Honavar, V.: On the semantics of linking and importing in modular ontologies. In: I. Cruz et al. (Eds.): ISWC 2006, LNCS 4273. (2006), 72-86.
6. Grau, B.C., Parsia, B., Sirin, E.: Working with multiple ontologies on the semantic Web. In: Proc. 3rd int. Semantic Web Conference (ISWC'2004). 620-634.
7. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M. E-connections of abstract descriptions systems; in Artificial Intelligence, vol.156, pages 1-73, 2004.
8. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A. Automatic Partitioning of OWL Ontologies Using E-Connections, in Proc. 2005 International Workshop on Description Logics (DL2005), CEUR-WS.org, vol. 147.
9. Stuckenschmidt, H., Klein, M. Structure-Based Partitioning of Large Concept Hierarchies, In Proc. 3rd Int. Semantic Web Conference (ISWC'2004), Springer , LNCS vol.3298, 289-303.
10. Devedzic, V. (2001). Knowledge Modeling - State of the Art. Integrated Computer-Aided Engineering 8, 257-281.
11. H. Stuckenschmidt and M. Klein, Structure based partitioning of large concept hierarchies, In Proc. 3rd Int. Semantic Web Conference, Hiroshima, Japan, (2004).
12. Core J2EE Patterns – Facade, http://java.sun.com/blueprints/corej2eepatterns/Patterns/SessionFacade.html.
13. Swoogle: A Search and Metadata Engine for the Semantic Web,http://swoogle.umbc.edu/.
14. Pizza Ontology, http://www.co-ode.org/ontologies/pizza/2007/02/12/
15.NeOn: Lifecycle Support for Networked Ontologies, Integrated Project, IST-2005-027595,http://www.neon-project.org/Web-content/.
16. Kaon2, http://kaon2.semanticWeb.org/.
17. Protégé, http://protege.stanford.edu/.
18. Sumo ontology, http://protege.stanford.edu/ontologies/sumoOntology/sumo_ontology.html.
19. P.Doran. Ontology Reuse via Ontology Modularisation. In Proceedings of KnowledgeWeb PhD, Symposium 2006 (KWEPSY2006). Budva, Montenegro. 17th June 2006.
20. Blomqvist, E.: State of the Art: Patterns in Ontology Engineering. Technical Report 04:8, Jönköping University, November 2004.
21. WonderWeb, http://wonderWeb.semanticWeb.org/.
22. Zimmermann, A., Euzenat., J.Three semantics for distributed systems and their relations with alignment composition. In Cruz, I., et al, eds, 5th Int.Semantic Web Conference, ISWC 2006,Athens, GA, USA, November 5-9, 2006, Proceedings, volume 4273 of Lecture Notes in Computer Science, pages 16–29. Springer-Verlag GmbH, November 2006.
23. Zhang, Mr. Yi and Vasconcelos, Dr. Wamberto and Sleeman, Prof. Derek (2004) OntoSearch: An Ontology Search Engine. In *Proceedings The Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI-2004)*, Cambridge, UK.
24. Gómez-Pérez, A., Fernández-López, M., Corcho, O. 2004. Ontological Engineering, Springer Verlag, Germany
25. Euzenat., J., Zimmermann, A., Freitas, F. Alignment-based modules for encapsulating ontologies, Proceedings of the Workshop on Modularization of Ontologies, Knowledge Capture Conference, Banff, Canada, 2007.
26. A. Farquhar, R. Fikes, & J. Rice. The Ontolingua Server: A Tool for Collaborative Ontology Construction. Knowledge Systems, AI Laboratory, 1996.
27. Paul Buitelaar, Thomas Eigner, Thierry Declerck OntoSelect: A Dynamic Ontology Library with Support for Ontology Selection In: Proc. of the Demo Session at the International Semantic Web Conference, Hiroshima, Japan, Nov. 2004.
28. d'Aquin, M., Baldassarre, M, Gridinoc, C, Angeletou, S and Motta, Enrico (2007-06), "WATSON: A Gateway for the Semantic Web", *ESWC 2007 poster session*.