3

Criteria and Evaluation for Ontology Modularization Techniques

Mathieu d'Aquin[1], Anne Schlicht[2],
Heiner Stuckenschmidt[2], and Marta Sabou[1]

[1] Knowledge Media Institute (KMi)
The Open University, Milton Keynes, UK
{m.daquin, r.m.sabou}@open.ac.uk
[2] University of Mannheim, Germany
{anne, heiner}@informatik.uni-mannheim.de

Summary. While many authors have argued for the benefits of applying principles of modularization to ontologies, there is not yet a common understanding of how modules are defined and what properties they should have. In the previous section, this question was addressed from a purely logical point of view. In this chapter, we take a broader view on possible criteria that can be used to determine the quality of a modules. Such criteria include logic-based, but also structural and application-dependent criteria, sometimes borrowing from related fields such as software engineering. We give an overview of possible criteria and identify a lack of application-dependent quality measures. We further report some modularization experiments and discuss the role of quality criteria and evaluation in the context of these experiments.

3.1 Introduction

Problems with large monolithical ontologies in terms of reusability, scalability and maintenance have lead to an increasing interest in techniques for extracting modules from ontologies. Currently, existing work suffers from the fact that the notion of modularization is not as well understood in the context of ontologies as it is in software engineering. While there is a clear need for ontology modularization, there are no well-defined and broadly accepted ideas about the criteria that define a good module. As a result, several approaches have been recently used to extract modules from ontologies, each of them implementing its own intuition about what a module should contain and what should be its qualities. In addition, a number of formal and informal modularization criteria have been proposed that are strongly influenced by certain use cases for modularization. This lack of consensus about quality criteria hinders the development of the field as a whole. On one hand, it is difficult to take up the results of the field outside itself because of a lack of guidelines about which technique to choose under which circumstances. On the other hand, within the field, it is impossible to compare the various techniques to each other.

Our hypothesis is that there is no universal way to modularize an ontology and that the choice of a particular technique or approach should be guided by the requirements of the application or scenario relying on modularization. In fact, we have already observed a strong

correspondence between the two major use cases in which modularization is needed and the two types of techniques that are used [7]. First, we distinguish scenarios where a large, monolithic ontology needs to be split up in order to allow its easier *maintenance and use* (e.g., by using reasoners and visualization tools). Accordingly, a significant group of techniques reported in the literature perform *ontology partitioning* by dividing an ontology into a set of significant modules that together form the original ontology [11, 6, 15]. The second class of scenarios, geared towards *selective use and reuse*, are those where a smaller part of an ontology that covers certain aspects of the domain is identified as a basis for a specific application. Candidate parts for reuse need to be small enough to be easily visualized and integrated in other applications than the one they have been initially built for. *Module extraction* techniques address such scenarios and refer to extracting a module from an ontology to be used for a particular purpose, i.e. covering a particular subvocabulary of the original ontology [14, 12, 7].

Based on the observation above, we believe that modularization criteria should be defined in terms of the applications for which the modules are created. In the remainder of this chapter, we survey existing criteria that can be used to evaluate modularization. Our goal is to provide a framework for evaluating and comparing modularization techniques according to application requirements, and so, a guideline to chose the right technique or combination of technique in a given scenario. Accordingly, we describe a set of experiments in which we apply a number of modularization techniques and analyze the results regarding the considered criteria. Also, as our main hypothesis is that the evaluation of modularization depends on the application requirements, these experiments are based a concrete applications scenario: the selection of relevant knowledge in existing ontology to be used in annotation.

The goal is to characterize the requirements of this particular application using the reviewed criteria and thus, to analyze the results of existing ontology modularization techniques regarding these requirements. Looking at the results of these experiments, we aim at better understanding the fundamental assumptions underlying the current modularization techniques and thus, at providing the building blocks for a more comprehensive evaluation, helping the application developers in choosing the appropriate technique and guiding the designers of techniques in further developments.

3.2 Use Cases for Modularization

The increasing awareness of the benefits of ontologies for information processing in open and weakly structured environments has lead to the creation of a number of such ontologies for real world domains. In complex domains such as medicine these ontologies can contain thousands of concepts. Examples of such large ontologies are the NCI Thesaurus with about 27.500 and the Gene Ontology with about 22.000 concepts. Other examples can be found in the area of e-commerce where product classification such as the UNSPSC or the NAICS contain thousands of product categories. While being useful for many applications, the size of these ontologies causes new problems that affect different steps of the ontology life cycle.

Maintenance:

Ontologies that contain thousands of concepts cannot be created and maintained by a single person. The broad coverage of such large ontologies normally requires a team of experts. In many cases these experts will be located in different organizations and will work on the same ontology in parallel. An example for such a situation is the Gene Ontology that is maintained by a consortium of experts.

Publication:

Large ontologies are mostly created to provide a standard model of a domain to be used by developers of individual solutions within that domain. While existing large ontologies often cover a complete domain, the providers of individual solutions are often only interested in a specific part of the overall domain. The UNSPSC classification for example contains categories for all kinds of products and services while the developers of an online computer shop will only be interested in those categories related to computer hardware and software.

Validation:

The nature of ontologies as reference models for a domain require a high degree of quality of the respective model. Representing a consensus model, it is also important to have proposed models validated by different experts. In the case of large ontologies it is often difficult, if not impossible, to understand the model as a whole due to cognitive limits. What is missing is an abstracted view on the overall model and its structure as well as the possibility to focus the inspection of a specific aspect.

Processing:

On a technical level, very large ontologies cause serious scalability problems. The complexity of reasoning about ontologies is well known to be critical even for smaller ontologies. In the presence of ontologies like the NCI Thesaurus, not only reasoning engines but also modelling and visualization tools reach their limits. Currently, there is no modelling tool that can provide convenient modelling support for ontologies of the size of the NCI ontology.

All these problems are a result of the fact that a large ontology is treated as a single monolithic model. Most problems would disappear, if the overall model consists of a set of coherent modules about a certain subtopic that can be used independently of the other modules while still containing information about its relation to these other modules.

3.3 Modularization Approaches

We consider an ontology O as a set of axioms (subclass, equivalence, instantiation, etc.) and the signature $Sig(O)$ of an ontology O as the set of entity names occurring in the axioms of O, i.e. its vocabulary.

In the following, we deal with several approaches for ontology modularization, having different assumptions about the definition of an ontology module. The assumption we adopt as a basis for our discussion is that a module is considered to be a significant and self-contained sub-part of an ontology. Therefore, an module $M_i(O)$ of an ontology O is also a set of axioms (an ontology), with the minimal constraint that $Sig(M_i(O)) \subseteq Sig(O)$. Note that, while it may often be desirable, it is not always the case that $M_i(O) \subseteq O$.

3.3.1 Ontology Partitioning Approaches

The task of partitioning an ontology is the process of splitting up the set of axioms into a set of modules $\{M_1, \cdots, M_k\}$ such that each M_i is an ontology and the union of all modules is

semantically equivalent to the original ontology O. Note that some approaches being labeled as *partitioning* methods do not actually create *partitions*, as the resulting modules may overlap. There are several approaches for ontology partitioning that have been developed for different purposes.

The approach of [11] aims at improving the efficiency of inference algorithms by localizing reasoning. For this purpose, this technique minimizes the shared language (i.e. the intersection of the signatures) of pairs of modules. A message passing algorithm for reasoning over the distributed ontology is proposed for implementing resolution-based inference in the separate modules. Completeness and correctness of some resolution strategies is preserved and others trade completeness for efficiency.

The approach of [6] partitions an ontology into a set of modules connected by $\mathcal{E}$-Connections. This approach aims at preserving the completeness of local reasoning within all created modules. This requirement is supposed to make the approach suitable for supporting selective use and reuse since every module can be exploited independently of the others.

A tool that produces sparsely connected modules of reduced size was presented in [15]. The goal of this approach is to support maintenance and use of very large ontologies by providing the possibility to individually inspect smaller parts of the ontology. The algorithm operates with a number of parameters that can be used to tune the result to the requirements of a given application.

3.3.2 Module Extraction Approaches

The task of module extraction consists in reducing an ontology to the sub-part, the module, that covers a particular sub-vocabulary. This task has been called segmentation in [14] and traversal view extraction in [12]. More precisely, given an ontology O and a set $SV \subseteq Sig(O)$ of terms from the ontology, a module extraction mechanism returns a module M_{SV}, supposed to be the relevant part of O that covers the sub-vocabulary SV ($Sig(M_{SV}) \supseteq SV$). Techniques for module extraction often rely on the so-called *traversal approach*: starting from the elements of the input sub-vocabulary, relations in the ontology are recursively "traversed" to gather relevant (i.e. related) elements to be included in the module.

Such a technique has been integrated in the PROMPT tool [12], to be used in the Protégé environment. This approach recursively follows the properties around a selected class of the ontology, until a given distance is reached. The user can exclude certain properties in order to adapt the result to the needs of the application.

The mechanism presented in [14] starts from a set of classes of the input ontology and extracts related elements on the basis of class subsumption and OWL restrictions. Some optional filters can also be activated to reduce the size of the resulting module. This technique has been implemented to be used in the Galen project and relies on the Galen Upper Ontology.

Inspired from the previously described techniques, [7] defines an approach for the purpose of the dynamic selection of relevant modules from online ontologies. The input sub-vocabulary can contain either classes, properties, or individuals. The mechanism is fully automatized, is designed to work with different kinds of ontologies (from simple taxonomies to rich and complex OWL ontologies) and relies on inferences during the modularization process.

3.4 Evaluation Criteria for Modularization

In the previous section, we have briefly presented a number of different approaches for ontology partitioning and module extraction. In this section, we take a closer look at different

criteria for evaluating either the modularization resulting from the application of a modularization technique or the system implementing the modularization technique. Before that, we start by looking at criteria that have been adopted for the classical notion of a module in software engineering.

3.4.1 Criteria from Software Engineering

The author of [9] reviews a set of features of software engineering modules with respect to what is called in this chapter the *requirements for logical modules*. From this analysis, two general criteria characterizing software engineering modules can be considered: encapsulation and independence.

Encapsulation.

Encapsulation refers to the distinction between the interface and the body (or implementation) of a program in software engineering. This distinction does not really apply when using Semantic Web technologies, but can be related to other notions like substitutability and information hiding. Indeed, the fact that a module can be easily exchanged for another, or internally modified, without side-effects on the application can be a good indication of the quality of the module. Module extraction techniques are intrinsically related to information hiding, since they aim at removing from the ontology the elements that are not related to the given subvocabulary, playing the role of an interface between the ontology and the application.

Independence.

A well-designed software module should be independent from the other modules used in the same applications in the sense that it should be reusable and exploitable separately. The same applies in ontology engineering, where ontology modules have to be self-contained and reusable components. Additionally, having independent modules is a way to improve the scalability of reasoning mechanisms by allowing to focus on a smaller set of elements (in the case of module extraction techniques, see e.g. [14]), or the use of distributed reasoning (in the case of partitioning techniques, see e.g. [11, 6]).

3.4.2 Evaluating Modularizations

Logical Criteria

If we look at ontologies as logical theories, it is a natural approach to define modularization criteria in terms of their logical properties, i.e. looking at their entailments. Recently, several papers have been interested in defining such criteria.

Local Correctness.

is probably the most obvious formal relation between a module $M_i(O)$ and its original ontology O. It states that every axiom being entailed by $M_i(O)$ should also be entailed by O. In other terms, nothing has been added in the module that was not originally in the ontology.

Local Completeness.

is the reverse property of local correctness. It is considered in many studies as the most important logical criterion concerning modularization. A module is said to be locally complete w.r.t. a local signature $Loc(M_i) \subseteq Sig(M_i)$ (e.g., the original set of entities of interest in an extraction technique) if every entailment of O that concerns only elements of $Loc(M_i)$ is preserved in $M_i(O)$. Local completeness has been formalized for example in [3] using the notion of *conservative extension* and also relates to the one of *uniform interpolant* described in [6].

Structural Criteria

[13] describes a set of criteria that can be computed on the basis of the structure of the modularized ontology. The criteria are inherently related to the previously mentioned software engineering criteria as they have been designed to trade-off maintainability as well as efficiency of reasoning in a distributed system, using distributed modules.

Size.

Despite its evident simplicity, the relative size of a module (number of classes, properties and individuals) compared to its source ontology is among the most important indicators of the efficiency of a modularization technique. Indeed, the size of a module has a strong influence on its maintainability and on the robustness of the applications relying on it: a big amount of information in one module leads to less flexibility in its exploitation and evolution. On the other hand, too small modules would not cover a sufficiently broad domain to be useful and would lead to problems related to other criteria (e.g. connectedness).

Intra-Module Distance.

It is worth to measure how the terms described in a module *move closer to each other* compared to the original ontology, as an indication of the simplification of the structure of the module. This *intra-module distance* is computed by counting the number of relations in the shortest path from one entity to the other. This is particularly relevant in the case of module extraction techniques, where reducing the distance between the input terms facilitates their joint visualization and helps in understanding their relationship in the original ontology.

Quality of the Modules

There are two different kinds of approaches for determining the quality of an existing ontology that can be used to evaluate the quality of ontology modules. While some approaches analyze the representational structures of an ontology on the basis of the actual content, others compare the content of the ontology with alternative representations of the domain (e.g. a document corpus) to determine how well it models relevant aspects of the domain it describes. We propose to use and combine these approaches in order to get an estimation of the suitability of a module for describing a certain aspect of a domain.

Module cohesion.

Cohesion denotes the degree of relatedness of elements within the same module. In the area of software engineering, a number of measures of cohesion have been defined that try to measure the connectedness of methods in a module based on criteria such as shared instance variables [2]. For the case of ontologies, Yao and his colleagues propose a set of cohesion metrics based on the structure of the inheritance hierarchy [17] and show that these metrics correlate with the intuition of human evaluators: the *number of root classes* in the hierarchy, the *number of leaf classes*, and the *maximum depth* of the hierarchy. The relevance of the definitions depends on all of these factors and only their combination provides a suitable indicator.

Richness of the representation.

Another important aspect related to the quality of modules is the amount of conceptual information retained in the module. We can measure this amount using some criteria that are inspired by the notion of schema richness proposed by Tatir and his colleagues [16] to measure the quality of ontologies. Here measuring the richness of the specifications in a module is based on the amount of relational information present in relation to the number of classes. We distinguish between the richness of the subsumption hierarchy – the average number of subclass relations per class – and the richness of the relations between classes – the average number of domain relations per class. It is clear that the richness of semantic information in a module strongly depends on the richness of the ontology the module originated from. A better indication of the quality is therefore provided by comparing the richness of the module with the richness of the corresponding set of concepts in the original ontology.

Domain coverage.

In the context of real applications domain coverage is the most important criterion as it determines how well the module fits the representational requirements of the application at hand. In order to be able to determine the domain coverage, we need a suitable representation of the domain that should be covered by the module. In cases where no instance data exists, we can adopt techniques of data-driven ontology evaluation that have been proposed in the area of ontology learning [1]. The idea is to compare the ontology with a corpus of documents in order to determine how well the ontology describes the content of the documents. These evaluations help in determining if the modularization technique have generated *significant* module according to the different domains or topics covered by the original ontology. What have to be evaluated is whether or not these modules are actually focused on a restricted number of domains, and if the domains are localized in the modularization, i.e. if they have not been spread over an important number of modules.

Relations Between Modules

Connectedness.

The independence (see Section 3.4.1), and so the efficiency, of a set of modules resulting from a partitioning technique can be estimated by looking at the degree of interconnectedness of the generated modules. A modularized ontology can be depicted as a graph, where the axioms are nodes and edges connect every two axioms that share a symbol. The connectedness of a module is then evaluated on the basis of the number of edges it shares with other modules.

Redundancy.

In case of partitioning techniques that allow modules to overlap, redundancy is a common way of improving efficiency and robustness. On the other hand, having to deal with redundant information increases the maintenance effort, and it has been shown in [11] that reasoning on non-redundant representations of parts of the complete model can lead to performance improvements.

In addition, computing the level of redundancy (the overlap) in modules generated using different techniques can be a way to compare and relate these techniques. More precisely, it can indicate whether these techniques rely on similar intuitions, if one is more general than the other, or, on the contrary, if they result in very different (and possibly complementary) modules.

Inter-module distance.

Counting the number of modules that have to be considered to relate two entities can help in characterizing the communication effort caused by the partition of an ontology. Indeed, if the modules resulting from an ontology partitioning technique are intended to be used on different machines, over a network, the *inter-module distance* gives an indication of the amount of distant access to other modules that is required to manipulate the considered entities.

3.4.3 Application Criteria

In [7] the authors focus on the use of modularization for a particular application. This leads to the definition of several criteria, most of them characterizing the adequacy of the design of a modularization tool with respect to constraints introduced by the application: assumptions on the ontology, the level of user interaction, and the availability of tools for using the resulting modules. Additionally, applications may have different requirements regarding the performance of the modularization tool.

Assumptions on the ontology.

Most of the existing approaches rely on some assumptions. For example, those described in [5] and [14] are explicitly made to work on OWL ontologies, whereas [15] can be used either on RDF or OWL but only exploits RDF features. In [14], the ontology is required to be well-designed and to use complex OWL constructs to describe classes. Moreover, some of the filters used to reduce the size of a module are dependent on elements of the Galen upper ontology.

Level of user interaction.

In many systems the required user entries are limited to the inputs of the algorithm. In certain cases, some numerical parameters can be required [15] or some additional procedures can be manually (de)activated [14]. The technique in [12] has been integrated in the Protégé ontology editor to support knowledge reuse during the building of a new ontology. In this case, modularization is an interactive process where the user has the possibility to extend the current module by choosing a new starting point for the traversal algorithm among the *boundary classes* of the module.

Use of modules.

Regarding the aspect of actually using modules in other applications, we only know of two approaches that make their modules available to reasoners/theorem provers (but not to any other applications). The modules extracted in [5] are linked together using $\mathcal{E}$-Connections and aim at being used in a reasoner. In a similar way, the knowledge base partitions created by the approache of [11] are used in a dedicated theorem prover.

Performance.

Most of the papers concerning modularization techniques do not give any indication about the performance of the employed method (with the noticeable exception of [14]). Performance is a particularly important element to be considered when using a modularization technique for the purpose of an application. Different applications may have different requirements, depending on whether the modularization is intended to be used dynamically, at run-time, or as a "batch" process.

3.5 Experiments with Modularization Techniques and Criteria

In this section, we apply the criteria described in the previous section to a particular application scenario, on the basis of two well defined examples. The idea is to evaluate how these criteria can be used in characterizing the application requirements and the assumptions underlying the modularization techniques.

3.5.1 The Knowledge Selection Scenario

Knowledge selection has been described in [7] as the process of selecting the relevant knowledge components from online available ontologies and has been in particular applied to the Magpie application. Magpie [8] is a Semantic Web browser which helps users to quickly make sense of the information provided by a Web page by allowing them to visually access the semantic data associated with that page. Available as a browser plugin in which a set of classes are displayed, Magpie can identify instances of these classes in the current Web page and highlight them with the color associated to each class. Core to Magpie is a manually selected ontology that contains the information needed to identify the relevant instances in Web pages.

In our current work we are extending Magpie towards open semantic browsing in which the tool automatically selects and combines online ontologies relevant to the content of the current page. As such, the user is relieved from manually choosing a suitable ontology every time he wishes to browse new content. Such an extension of our tool relies on mechanisms that can not only dynamically select appropriate ontologies from the Web, but can also extract from these ontologies the relevant and useful parts to describe classes in the current Web page.

Our previous work and experiences in ontology selection [10] made it clear that modularization may play a crucial role in complementing the current selection techniques. Indeed, selection algorithms tend to run into two major problems. First, if the selection returns a large ontology this is virtually useless for a tool such as Magpie which only visualises a relatively small number of classes at a time. Unfortunately, in the experiments we have performed large ontologies are often returned. What is needed instead is that the selection process returns a part (module) of the ontology that defines the relevant set of terms. A second problem is that

in many cases it is difficult to find a single ontology that covers all terms (we observed this knowledge sparseness phenomenon in [10]). However, a combination of one or more ontologies could cover all the query terms. This problem is related to modularization in the sense that it is easier to combine small and focused knowledge modules than ontologies of large size and coverage.

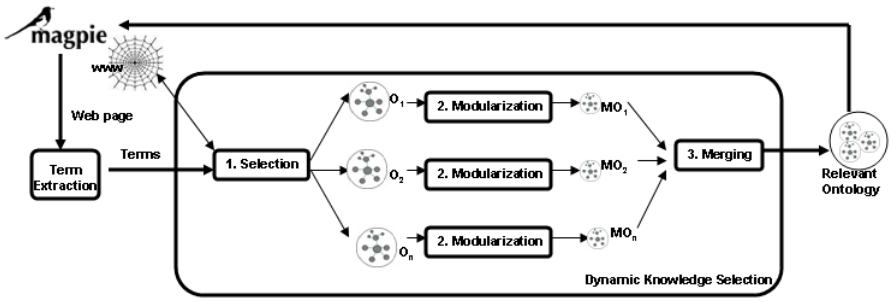

Fig. 3.1. The knowledge selection process and its use for semantic browsing with Magpie.

These considerations justify the need to extend selection techniques with modularization capabilities. In Figure 3.1 we depict the three major generic steps of the *knowledge selection* process that integrates ontology selection, modularization and merging.

3.5.2 Experimental Setting

In the scenario described in the previous section, modularization is integrated in a fully automatic process, manipulating automatically selected online ontologies for the purpose of annotation in Magpie. In this section, we simulate the process of knowledge selection on two examples, using four different techniques, in order to evaluate and compare their results[3]. The purpose is to characterize the requirements of this particular scenario using the criteria defined in Section 3.4, and to show how modularization techniques respond to the selected experiments regarding these requirements.

The Examples

We consider two examples, originally described in the context of ontology selection in [10], where the goal is to obtain an ontology module for the annotation of news stories. We simulate the scenario described in Section 3.5.1 by manually extracting relevant keywords in these stories, using ontology selection tools[4] to retrieve ontologies covering these terms, and then applying modularization techniques on these ontologies (steps 1 and 2 in figure 3.1).

In the first example, we consider the case where we want to annotate the news stories available on the KMi website[5]. We used the keywords *Student*, *Researcher*, and *University* to select ontologies to be modularized, and obtain three ontologies covering these terms:

[3] Actual results are available at http://webrum.uni-mannheim.de/math/lski/Modularization

[4] in particular Watson (http://watson.kmi.open.ac.uk).

[5] http://news.kmi.open.ac.uk/

ISWC: http://annotation.semanticweb.org/iswc/iswc.owl
KA: http://protege.stanford.edu/plugins/owl/owl-library/ka.owl
Portal: http://www.aktors.org/ontology/portal

It is worth to mention that this example is designed to be simple: we have chosen a well covered domain and obtained three well defined OWL ontologies of small size (33 to 169 classes).

The second example was used in [10] to illustrate the difficulties encountered by ontology selection algorithms. Consequently, it also introduces more difficulties for the modularization techniques, in particular because of the variety of the retrieved ontologies in terms of size and quality. It is based on the following news snippet:

"The Queen will be 80 on 21 April and she is celebrating her birthday with a family dinner hosted by Prince Charles at Windsor Castle"[6]

Using the keywords *Queen*, *Birthday* and *Dinner*, we obtained the following ontologies, covering (sometimes only partially) this set of terms:

OntoSem: http://morpheus.cs.umbc.edu/aks1/ontosem.owl
TAP: http://athena.ics.forth.gr:9090/RDF/VRP/Examples/tap.rdf
Mid-Level: http://reliant.teknowledge.com/DAML/Mid-level-ontology.owl, covering only the terms *Queen* and *Birthday*

Compared to Example 1, the ontologies used in Example 2 are bigger (from 1835 classes in **Mid-Level** to 7596 in **OntoSem**). Moreover, they contain different levels of descriptions. For example, **OntoSem** is a big, complex OWL ontology containing a lot of properties (about 600), whereas TAP is simple RDFS taxonomy without any properties. In that sense, we use Example 1 to assess basic characteristics of the modularization techniques and then, rely on Example 2 to show how these characteristics are influenced by the properties of the ontologies.

Evaluated criteria.

Logical criteria are of particular importance when the modules resulting of the modularization techniques are intended to be used in reasoning mechanisms, but should not be emphasized in our use case, which focuses on a human interpretation of the module.

On the contrary, structural criteria are fundamental indicators for estimating the efficiency of the modularization techniques. Indeed, the size of the returned ontology module is crucial since Magpie only needs relevant parts of ontologies, small enough to be visualized within the browser and to be easily interpreted by the user in relation with the current Web page. However, it is essential to keep enough knowledge in the module to maintain a understandable structure around the considered terms.

Evaluating the quality of the resulting modules is a crucial task, in particular in applications where modularization is intended to facilitate ontology reuse. However, applying and interpreting the metrics for evaluating module quality require important (human) effort, which go beyond the scope of our experiments, intending to simulate an automatic process.

Criteria dedicated to sets of interconnected modules resulting from partitioning techniques – *redundancy*, *connectedness*, and *inter-module distance* – are not relevant in the considered scenario, as we are only interested in one module. One of the goals of modularization in our use case is to facilitate the exploitation, and so the interpretation, of the knowledge related to the input terms in ontologies. In that sense, the *intra-module distance* between these terms

[6] http://news.billinge.com/1/hi/entertainment/4820796.stm

should be reduced in such a way that they can be considered and apprehended together by the user.

In the knowledge selection scenario, modularization is integrated in a complete process, leading to particular constraints concerning application criteria. The results of Example 2 should help to better understand which *assumptions* the different techniques rely on. Since knowledge selection is fully automatized, the required level of *user interaction* should be minimal. Finally, in the considered scenario, modularization is intended to be used at runtime, leading to fundamental constraints concerning the *performance* of the modularization tool.

3.5.3 Experimented Techniques

As already described in [7], it is quite obvious that module extraction techniques fit better in the considered scenario than partitioning tools. Indeed, we want to obtain *one* module covering the set of keywords used for the selection of the ontology and constituting a *sub-vocabulary* of this ontology. However, the result of partitioning techniques can also be used by selecting the set of generated modules that cover the considered terms. The criteria are then evaluated on this set of modules as grouped together by union.

We have chosen to consider only available partitioning and module extraction techniques that are sufficiently stable and that can easily be used on the previously described ontologies (e.g., without requiring language conversion). We have selected two ontology partitioning tools:

PATO: A standalone application described in [15].

SWOOP: The partitioning functionality included in the SWOOP ontology editor and described in [5].

and two module extraction tools:

KMi: A standalone application developed at KMi for the purpose of the knowledge selection scenario, as described in [7].

Prompt: The module extraction feature of the Prompt toolkit, integrated as a plugin of the Protégé ontology editor, as described in [12].

These tools are described in more details below.

Ontology Partitioning Technique: SWOOP

SWOOP[7] is a popular ontology editor, focused on OWL ontologies and relying on an hypertext-like way of navigating in the ontology entities (see Figure 3.2). SWOOP is developed by the same group who made the Pellet OWL reasoner, and inference capabilities can be used during the editing process thanks to Pellet. Another particularity of SWOOP is that, in addition to standard OWL ontologies, it can manage $\mathcal{E}$-Connections based ontologies: local ontologies linked together by $\mathcal{E}$-Connections [4].

SWOOP integrates a fully automatic ontology partitioning functionality. This functionality, based on the paper [5], is supposed to divide standard OWL ontologies to create a set of local ontologies, linked together by $\mathcal{E}$-Connections. This partitioning feature of the SWOOP editor is further referred to as **SWOOP**.

From these elements we can already get an evaluation of some of the *application criteria*: **SWOOP** is fully automatic, works only on OWL ontologies (without imports) and there exist tools (the SWOOP editor, Pellet) to exploit the resulting set of modules.

[7] http://www.mindswap.org/2004/SWOOP/

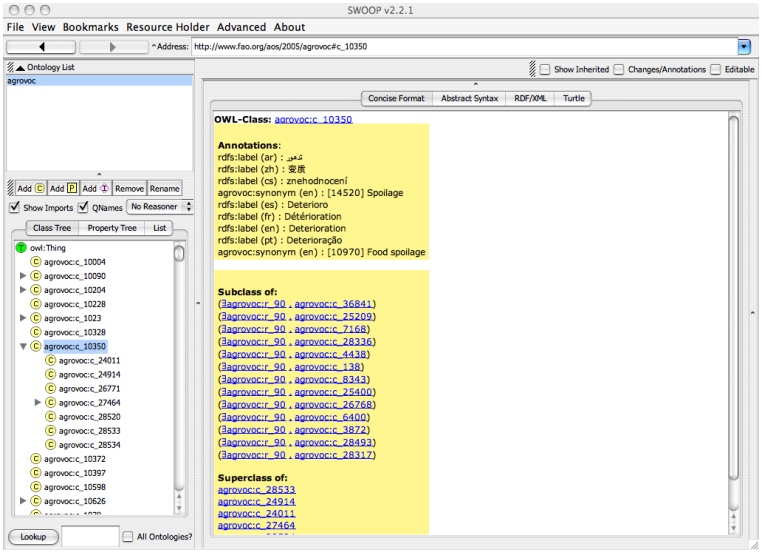

Fig. 3.2. Screenshot of the SWOOP editor.

Ontology Partitioning Technique: PATO

PATO is a standalone application written in java (see Figure 3.3) and implementing the technique described in [15]. In principle, **PATO** divides an ontology into a set of modules by first computing a *dependency graph* between the entities of the ontology, relying on RDF(S) relations between them. Entities are then clustered according to measures inspired from the field of network analysis, aiming at minimizing the interconnections between modules.

Concerning the application criteria, as it can be seen in Figure 3.3, **PATO** is supposed to be fine-tuned using different parameters at each step of the partitioning process. However, an ongoing work is currently conducted towards the automatic configuration of **PATO** [13]. No particular reasoner is available for exploiting modules resulting from this tools. However, each module is a self-contained OWL ontology that can be used with usual tools. Finally, even if it works with most of the ontologies in RDF(S) or OWL, **PATO** only exploits the representation primitive of RDF(S).

Module Extraction Technique: Prompt

Prompt[8] is a toolkit, integrated as a plugin of the Protégé ontology editor[9] for comparing, merging and extracting views from ontologies. What we refer to as **Prompt** in the following corresponds to the part that concerns module extraction and that is called *traversal view extraction* in the Prompt toolkit [12]. This tool is designed as an interactive process for extracting sub-parts (modules, views) of ontologies to be integrated in the currently edited ontology (see

[8] http://protege.stanford.edu/plugins/prompt/prompt.html
[9] http://protege.stanford.edu/

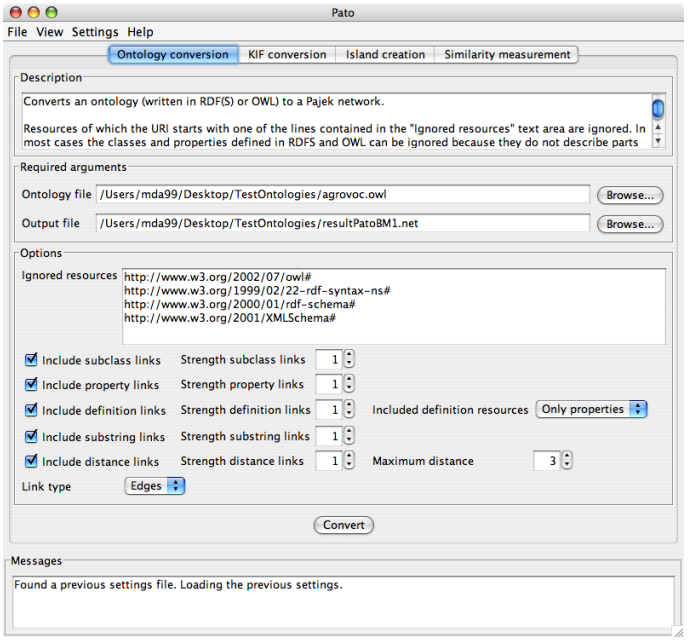

Fig. 3.3. Screenshot of the **PATO** tool.

Figure 3.4). A class of the source ontology is first selected by the user, using the standard navigation interface of Protégé. The principle of the approach is to recursively "traverse" the ontology relations from this class to reach other classes to be included. The relations to follow and the distances for which they have to be followed (the level of recursion) have to be first entered by the user. The whole process is incremental in the sense that new classes can be selected from the boundaries of the current module as new starting points for the extraction of other classes, which are added to the module, expending its boundaries.

Concerning the criterion on the *level of user interaction*, it is quite obvious that **Prompt** is not automatic at all: it requires the intervention of the user at each step of the process. The interactive nature of **Prompt** make it harder to evaluate (the resulting module is dependent on the user who defined it), but can also be seen as an advantage, as **Prompt** is less dependent on a particular intuition on modularization, and so, less restricted in terms of usage scenarios. Concerning other application criteria, **Prompt** can be used on any ontology manageable by Protégé and generates modules (views) that are integrated in Protégé ontologies. The performance of the system in terms of time is obviously an irrelevant criterion for evaluating **Prompt**.

Module Extraction Technique: KMi

What is called here **KMi** is a technique developed at the Knowledge Media Institute (the Open University, UK) and described in [7]. It is also a "traversal approach", inspired from the two previous techniques for module extraction. One of the particularities of **KMi** is that it takes

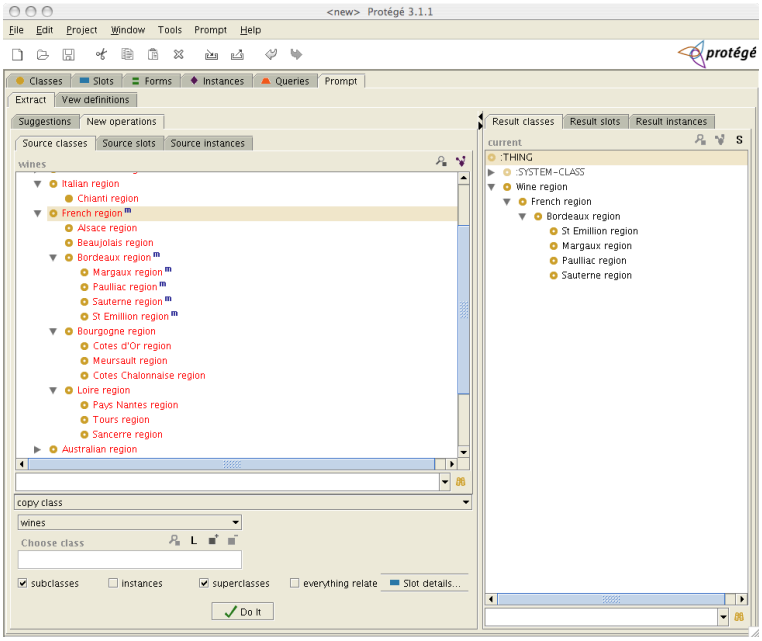

Fig. 3.4. Screenshot of the **Prompt** view extraction tool.

into account inferences during the modularization process, in order to validate some interesting logical properties. Moreover, **KMi** generally generates smaller modules (e.g. than **GALEN**) by taking shortcuts in the ontology hierarchy[10], while keeping all the necessary elements for describing the included entities.

KMi takes as an input only a sub-vocabulary of the source ontology, in the form of a set of class, property and individual names. Some other parameters can be modified through the interface (see Figure 3.5) but, since only the sub-vocabulary is required, **KMi** can be considered as fully automatic.

Moreover, **KMi** has been designed to work with – and exploit the content of – any kind of ontology, from simple taxonomies in RDF(S) to complex OWL structures. Finally, as the resulting module is a standard, self-contained ontology, there is no need for particular tools to exploit it.

3.5.4 Results

Running the four modularization techniques on the three ontologies of the first example allowed us to test how they behave on simple, but yet practical real word examples. The second example concerns larger ontologies, with more heterogeneous levels of description. For example, **TAP** contains around 5500 classes, but no property or individual, whereas **Mid-Level**

[10] In particular, by including only the common super class of included classes, rather than the whole branches of super-classes.

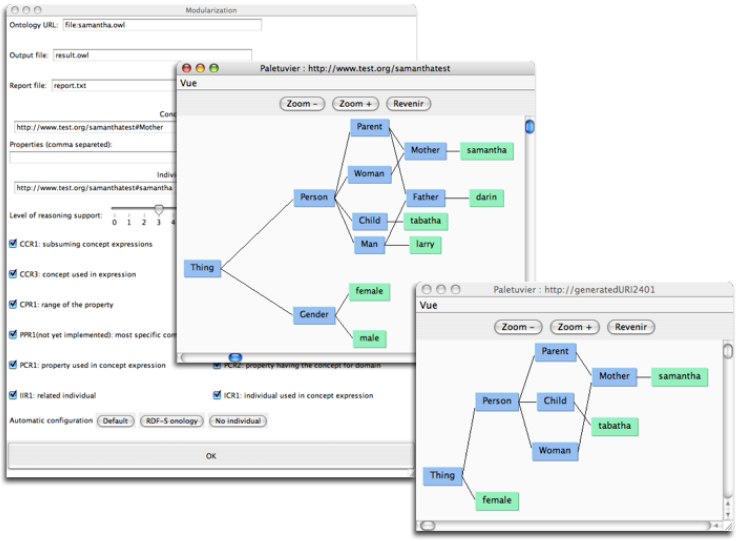

Fig. 3.5. Screenshot of the **KMi** module extraction tool.

relies on almost 200 properties and is populated with more than 650 individuals for less than 2000 classes.

Evaluating the Modularizations

Analyzing the modules resulting from the considered modularization techniques is a way to better understand on which kinds of assumptions these techniques rely, and if these assumptions fit the requirements of the application.

Size.

Figure 3.6 shows the *size* of the resulting modules for each system on Example 1 in terms of number of classes and properties (we did not include number of individuals as it is not a relevant indicator here). It can be easily remarked that **SWOOP** generally generates very large modules, containing 100% of the classes for two of the three ontologies, and an important proportion of the properties: in most of the cases, **SWOOP** generates one module with almost the same content as the original ontology. Because it has not been really configured for the experiment, **Prompt** also generates big modules. The tool developed in **KMi** is focused on generating modules with a small number of classes (the smallest), so that the ontology hierarchy would be easy to visualize. It nevertheless includes a large proportion of the properties, in order to keep the definition of the included classes intact. **Pato** is optimized to give an appropriate size. It generally operates an important reduction of the size of the ontology.

Concerning Example 2, the difference between **SWOOP** and other techniques is even more significant. Indeed, because of the poor structure of the considered ontologies (restricted

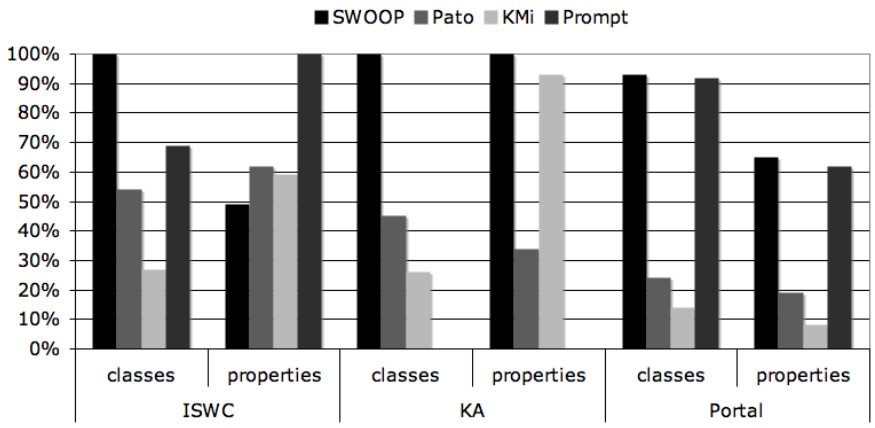

Fig. 3.6. Relative size of the resulting modules for the first example.

uses of OWL constructs, few or insufficiently defined properties), **KMi** and **Pato** result in particularly small modules (less than 10 classes), whereas **SWOOP** still includes most of the content of the ontology in a single module.

Intra-module distance.

The **KMi** tool relies on mechanisms that "takes shortcuts" in the class hierarchy for reducing the size of the module. Indeed, instead of including all the super-classes of the included classes, it only considers classes that relate these entities: their common super-classes. In that sense, the *distance* between the considered terms is also reduced in modules provided by **KMi**. For example, in the **Portal** ontology, by eliminating an intermediary class between *Researcher* and *Person*, **KMi** has reduced the distance between *Researcher* and *Student*, while keeping a well formed structure for the module. Since they do not include this kind of mechanisms, the other techniques generate modules in which the distance between included terms are the same as in the original ontology.

Logical criteria.

SWOOP is the only tool that can guarantee the *local completeness* of the module. The focus **SWOOP** put on logical criteria can be an explanation of its unsatisfactory results concerning the size of the module. **KMi** has been designed to provide modules where a weaker notion of local completeness holds [7], which can be useful to facilitate the interpretation of the module by the user. This property generally holds also for other techniques.

Application Criteria

Application criteria takes an important part of the evaluation, as our goal is to integrate the modularization technique into an broader scenario having particular requirements. Table 3.1 summaries the evaluation of these criteria on the considered tools.

	Ontology	Interaction	Tool support	Perf. (Ex1/Ex2)
SWOOP	RDF(S)/OWL (in SWOOP)	Automatic	SWOOP/ Pellet	(few sec/sec-min)
PATO	RDF(S)	Parameters	OWL tools	(few sec/min)
Prompt	RDF(S)/OWL, Frame (in Protégé)	Interactive	Protégé	N/A
KMI	RDF(S)/OWL (parsed by Jena)	Automatic	OWL tools	(few sec/min)

Table 3.1. Evaluation of the application criteria: Assumption on the source ontology, level of required user interaction, availability of tools for manipulating modules, and performance on both Example 1 and Example 2.

Level of interaction.

As already mentioned, **SWOOP** is fully automatic and does not need any parameters besides the input ontology. As a module extraction tool, **KMi** requires, in addition to the source ontology, a set of terms from the signature of the ontology, defining the sub-vocabulary to be covered by the module. This sub-vocabulary corresponds to the initial terms used for selecting the ontology: *Researcher*, *Student* and *University*. **Pato** has to be fine tuned with several parameters, depending on the ontology and on the requirements of the application. Here, it has been configured in such a way that modularizations in which the considered terms are in the same module are preferred. **Prompt** is an interactive mechanism, in which the user is involved in each step of the process. In particular, the class to be covered and the property to traverse have to be manually selected, requiring that the user has a good insight of the content of the ontology, can easily navigate in it, and that he understands the modularization mechanism. When using **Prompt**, we manually included the input terms and tried to obtain an (intuitively) good module, without going too deep in the configuration. Note that, since the system crashed at the early stage of the process, we did not manage to obtain results for the **KA** ontology with **Prompt**.

Assumption on the source ontology.

In addition to what have been already mentioned concerning the type of ontologies the modularization tools are supposed to handle, it follows from the experiments that some of the techniques are not designed to take into account big and heterogeneous ontologies like the ones of Example 2. It is particularly hard for the user to handle the process of module extraction in **Prompt** when having to deal with several thousands of classes and hundreds of properties. We also did not manage to partition the **OntoSem** ontology using **Pato** because of the way **OntoSem** makes use of the `label` annotation property.

Moreover, the results obtained concerning the size of the modules in Example 2 shows that techniques are highly influenced by the inherent properties of the ontology to be modularized and that, in general, they *assume* a high level of description.

Performance.

Apart from **Prompt** for which this criteria is irrelevant, each tool has only taken a few seconds or less on the *small ontologies* of Example 1. The ontologies in Example 2 are bigger and

more heterogeneous. These elements obviously have an important impact on the performance of the modularization techniques: in the worst cases (**Pato** and **KMi** on **TAP**), it takes several minutes to get a modularization and none of the tested techniques can be used at run-time for such ontologies. However, as already observed in [14], loading and processing a big ontology generally takes longer than the actual modularization process.

3.6 Conclusion and Discussion

There is currently an important growth in interest concerning modularization techniques for ontologies, as more ontology designers and users become aware of the difficulty of reusing, exploiting and maintaining big, monolithic ontologies. The considered notion of modularity comes from software engineering, but, unfortunately, it is not yet as well understood and used in the context of ontology design as it is for software development. Different techniques implicitly rely on different assumptions about modularity in ontologies and these different *intuitions* require to be made explicit.

We have reported on preliminary steps towards the characterization of ontology modularization techniques. We reviewed existing modularization tools as well as criteria for evaluating different aspects of a modularization (logical, structural, application level), and used them on a particular scenario: the automatic selection of knowledge components for the annotation of Web pages. The main conclusion of these experiments is that the evaluation of a modularization (technique) is a difficult and subjective task that requires a formal, well described framework – a *benchmark* – taking into account the requirements of applications. Such a framework would be useful in two ways: first for application developers, it would provide a guide for choosing the appropriate modularization technique, and second, for the developers of modularization techniques, it would give directions in which techniques can be improved with respect to particular scenarios. More detailed conclusions are presented below, focusing on issues to be addressed for the evaluation of modularization techniques.

No Universal Modularization. As described in [7], the technique developed at **KMi** has been explicitly designed for the purpose of the knowledge selection scenario. Therefore, it is not really surprising that it obtained almost the best results for most of the evaluated criteria. Beyond the simple comparison of techniques, this result tends to demonstrate our original assumption: the evaluation of a modularization depends on the application requirements. Indeed, other scenarios may require more logical properties to hold, a better defined distribution of the module, or a more active involvement of the user, leading to the use of other techniques. It appears that techniques are designed to be used in particular scenarios, and so, that it is important to characterize them in terms of the requirements they fulfill to facilitate the development of applications relying on ontology modularization.

Existing Criteria are not Sufficient. In our experiments, we rely on criteria that have been explicitly used to evaluate different aspects of a modularization, with the underlying assumption that these aspects are relevant indicators of its efficiency, its usability or its relevance. However, when looking at the resulting modules, it seems obvious that important criteria are missing for evaluating the quality of a modularization. For example, in our scenario, it seems fundamental that the module keeps a *good structure* or that it maintains a *well defined* description of the included entities. It can be argued that it is the role of *logical criteria* to evaluate how formal properties are preserved in ontology modules, but it can be easily shown (at least experimentally) that they are insufficient to evaluate the *design quality* of a modularization. Integrating the evaluation of the quality (or rather qualities) of the produced modules is therefor an important task. We cannot expect modularization techniques to generate good quality

modules from poorly designed ontologies, but, using these metrics, we can measure to which extent the inherent qualities of the source ontology are preserved in its modularization.

Techniques Need to be Improved. The difference in quality of the results for our second example (big, heterogeneous ontologies), compared to the first one (small, well defined ontologies), shows that, even if they are generally well designed and implemented, ontology modularization techniques need lots of improvements in terms of robustness, stability and scalability to be actually usable in real life scenarios. The evaluation of the considered criteria, taking into account different aspects of modularization, explicitly demonstrates that existing modularization techniques rely on different assumptions on ontologies and modularity of ontologies. Being restricted to particular use cases and ontologies prevent these techniques to be really usable in an environment like the Semantic Web, considering its inherent scale and heterogeneity.

References

1. C. Brewster, H. Alani, S. Dasmahapatra, and Y. Wilks. Data Driven Ontology Evaluation. In *Proceedings of International Conference on Language Resources and Evaluation*, 2004.
2. S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.
3. B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. A Logical Framework for Modularity of Ontologies. In *Proc. of the International Joint Conference on Artificial Intelligence, IJCAI*, 2007.
4. B. Cuenca Grau, B. Parsia, and E. Sirin. Combining OWL ontologies using E-Connections. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(1):40–59, January 2006.
5. B. Cuenca Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Automatic Partitioning of OWL Ontologies Using E-Connections. In *Proc. of Description Logic Workshop (DL)*, 2005.
6. B. Cuenca Grau, B. Parsia, E. Sirin, and A Kalyanpur. Modularity and Web Ontologies. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning, KR*, 2006.
7. M. d'Aquin, M. Sabou, and E. Motta. Modularization: a Key for the Dynamic Selection of Relevant Knowledge Components. In *Proc. of the ISWC 2006 Workshop on Modular Ontologies*, 2006.
8. M. Dzbor, J. Domingue, and E. Motta. Magpie - towards a semantic web browser. In *Proc. of the Second International Semantic Web Conference (ISWC)*, 2003.
9. F. Loebe. Requirements for Logical Modules. In *Proc. of the ISWC 2006 Workshop on Modular Ontologies*, 2006.
10. V. Lopez M. Sabou and E. Motta. Ontology Selection on the Real Semantic Web: How to Cover the Queens Birthday Dinner? In *Proc. of the European Knowledge Acquisition Workshop (EKAW)*, Podebrady, Czech Republic, 2nd-6th October 2006.
11. B. MacCartney, S. McIlraith, E. Amir, and T.E. Uribe. Practical Partition-Based Theorem Proving for Large Knowledge Bases. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.
12. N.F. Noy and M.A. Musen. Specifying Ontology Views by Traversal. In *Proc. of the International Semantic Web Conference (ISWC)*, 2004.
13. A. Schlicht and H. Stuckenschmidt. Towards Structural Criteria for Ontology Modularization. In *Proc. of the ISWC 2006 Workshop on Modular Ontologies*, 2006.

14. J. Seidenberg and A. Rector. Web Ontology Segmentation: Analysis, Classification and Use. In *Proc. of the World Wide Web Conference (WWW)*, 2006.
15. H. Stuckenschmidt and M. Klein. Structure-Based Partitioning of of Large Concept Hierarchies. In *Proc. of the International Semantic Web Conference (ISWC)*, 2004.
16. S. Tartir, I. Budak Arpinar, M. Moore, A. P. Sheth, and B. Aleman-Meza. OntoQA: Metric-Based Ontology Quality Analysis. In *IEEE ICDM 2005 Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 2005.
17. H. Yao, A. M. Orme, and L. Etzkorn. Cohesion metrics for ontology design and application. *Journal of Computer Science*, 1:107–113, 2005.