A Flexible Partitioning Tool for Large Ontologies

Anne Schlicht, Heiner Stuckenschmidt
KR and KM Research Group
University of Mannheim
A5, 6 68159 Mannheim, Germany
{anne, heiner}@informatik.uni-mannheim.de

Abstract

The benefits of modular ontologies in terms of easier creation and maintenance as well as better computational properties have been recognized by different researchers. As most real world ontologies, however, are still designed in a monolithic way, there is a need for methods that partition an existing ontology into a set of modules. 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. In this paper we present a flexible partitioning tool for large ontologies that can be adapted to the needs of different applications based on criteria that the resulting modular ontology should satisfy.

1 Introduction

With the increasing use of ontologies in many branches of science and industry not only the number of available ontologies has increased considerably but also many widely used ontologies have reached a size that overburdens development and quality control procedures. It has been argued that the maintenance of large ontologies would be greatly facilitated by decomposing large ontologies into smaller modules [10] that cover certain subtopics of the ontology. Using such models it would be much easier to avoid inconsistencies and semantic defects in the creation and maintenance when the part of the ontology to consider for modification is clearly defined [6].

While there is a clear need for 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 partition ontologies into modules, each of them implementing its own intuition about what a module should contain and what should be its qualities [5, 2, 10].

Finding an appropriate modularization tool for a given application is difficult. Testing candidate tools with the application is time consuming, as the characteristics of different modularizations have to be compared manually. Furthermore, some applications requirements are not met by any available tool. We address this problem by proposing a flexible modularization tool that can be adapted to requirements of an application and gives comprehensive support for automatical configuration.

1.1 Outline and Contribution

In the following, we investigate the partitioning of large ontologies into a set of interconnected modules based on structural quality criteria using the PATO tool. The approach is based on previous work on structure-based ontology partitioning reported in [10]. We extend this work in following way:

1. We present a method for automatically selecting optimal parameters for the algorithm that maximize the quality of the result. The method is an extension of the work that was briefly presented in [7].
2. We show that the criteria can be adapted according to application needs using two scenarios: a distributed reasoning application that requires an ontology modularization that minimizes communication cost and a visualization application that requires the reduction of the ontology to a small set of comprehensive modules.

All methods described in this paper have been implemented in the PATO system, a tool for partitioning OWL ontologies that is available online[1].
The paper is structured as follows. In the next section, we briefly present a revised version of the structure-based partitioning algorithm from [10]. In section 3 we propose a set of graph measures that can be used as structural quality criteria for modular ontologies. Section 4 discusses the problem of automatic parameter optimization based on the

[1] http://webrum.uni-mannheim.de/math/lski/Modularization/pato.html

criteria presented in section 3. In section 5, we show results of applying the partitioning method to real world ontologies according to different criteria. We conclude with a discussion of the approach and future work.

2 Partitioning Algorithm

In the following, we review the structure-based partitioning algorithm proposed in [10, 9] and describe a number of optimizations developed in the meantime. The resulting enhanced partitioning algorithm provides the basis for the work on criteria-driven ontology partitioning described in the subsequent sections.

2.1 Basic Method

The basis for the work reported in this paper is the partitioning method presented in [10]. It consists of a basic algorithm for splitting up ontologies based on an encoding of dependencies between elements in the ontology in a graph structure and finding sets of strongly related nodes. To this basic method we added *partition realization*, i.e. the creation of a distributed ontology from the partitioned set of ontology elements. Before we turn to the criteria-based optimization, we briefly describe the three steps of the partitioning method: First a graph is created from the ontology, second this graph is partitioned and third the partitioning is realized by distributing the ontology.

Step 1: Create Dependency Graph: In the first step, a graph structure is created that represents the dependencies between elements in the ontologies. For RDF and OWL ontologies, nodes in the graph are values of "rdf:label" or "rdf:ID" depending on the configuration. Edges can be created for subclass relations, classes that use the same property, terms that appear in the same definition and similar names (based on substring and edit distance). Details can be found in [11].

Step 2: Graph Partitioning In the second step, the notion of a line island [1] (a set of nodes of given minimal and maximal size for which the strength of the connection between the nodes inside the set is higher than the strength of any connection to nodes outside the set) is used to determine sets of ontology elements that should be in one module.

Step 3: Partition Realization Finally, a distributed ontology is created based on the graph partitioning. In particular, a namespace is created for each ontology module and the namespace of each ontology element is changed accordingly. Depending on the users preferences, the algorithm outputs either a set of ontology files or one ontology file with adapted namespaces. If redundancies in the distributed representation of the ontology are allowed, each axiom may be copied to different modules to increase the degree of independency between the modules.

Note that each step of the algorithm depends on parameters that can be set to adapt the resulting partitioning to given requirements. Examples include the following:

- In step 1, it has to be decided on what basis links between nodes in the dependency graph are created. Normally this means deciding between the options mentioned in the description above. In principle, these options can also be combined.
- In step 2, the most important parameter is the maximal size of a line island that determines how many nodes are allowed to end up in the same module. Reducing this parameter forces the algorithm to split up more tightly connected nodes.
- In step 3, the maximal amount of redundancy can be used to determine the amount of axioms that are copied to other modules to reduce interactions

The optimal choice of these parameters heavily depends on the applications and more specifically on the criteria that determine the quality of the resulting partitioning with respect to that application. The core idea of this work is to determine the optimal parameter set based on information about quality criteria for a given application of the partitioning method. We discuss these quality criteria in the following section.

3 Structural Quality Criteria

In order to judge whether a given modularization is a good one with respect to the goals of the distribution, we define a number of structural quality criteria that indicate whether a given modularization is likely to be adequate for a given scenario. We base these criteria on the following abstract model of an ontology: we consider an ontology to be a set of axioms $\{A_1, \cdots, A_n\}$. These axioms can for instance be rules, concepts definitions in description logics or axioms in any other logical language. We consider the number of axioms n to be the size of the ontology. The task of partitioning an ontology $\mathcal{O}$ can now be described as 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
- $\bigcup_{i=1}^{k} M_i = \mathcal{O}$

This definition leaves room for a large variety of different ontology languages and usage scenarios.

3.1 Size Distribution

The first set of criteria implemented in the PATO Tool addresses the assignmnet of axioms to the modules. These are standard graph measures, that can be used and combined to form new criteria.

Number of Modules The most obvious criterion of a partitioning is the number of modules it consists of.

Average Module Size If the partitioning is non-redundant, the average module size is the size of the ontology divided by the number of modules.

Variance of Sizes The variance is the sum of the squares of the derivations of each module size from the average size.

Size Balance Balance is computed by dividing the size of the largest module by the average size. If this value is close to 1, the sizes are well balanced.

While these criteria might be sufficient in some cases, for instance if the goal is to split up the ontology into a fixed number of modules with a maximal size for visualization purposes, many applications ask for criteria that cannot be determined on the basis of the assignment alone.

3.2 Links Between Ontology Elements

Based on implicit or explicit relations between elements in different modules (i.e. relations such as the ones encoded in the dependency graph created in step 1 of the partitioning algorithm) additional graph measures are applied.

We define the *module graph* to be obtained from the dependency graph and allocation of nodes (i.e. ontology elements) to modules by merging nodes allocated to the same module. Edges with identical source and target node are ignored. The resulting graph can be used as the basis for defining additional criteria such as the following.

Edge Cut The weighted sum of the edges in the module graph.

Relative Edge Cut Dividing the edge cut by the weighted sum of the edges in the dependency graph gives the relative edge cut.

Cut Balance Similar to the size balance, edge balance is the highest occurring weight in the module graph divided by the average weight.

These criteria are also based on standard notions in graph theory, depending on the nature of the edges in the module graph, however, they become meaningful with respect to certain applications. In a distributed reasoning scenario, for example, edges representing overlap of terms in different axioms indicate the need for communication between modules and therefore provide an adequate basis for minimizing communication costs by a suitable partitioning.

3.3 User Defined Criteria

Because we cannot foresee all possible applications where partitioning will be used and the corresponding criteria adequate for the respective application, the ability to define new quality criteria is needed. We support the easy integration of such user defined criteria by providing a simple Interface for plugging in new criteria. The user only needs to implement a Java interface that contains two methods, one for setting the input and one for getting the result. If the name of the interface implementation and the new criteria is declared in the configuration file, the new class is used automatically without recompiling PATO.

4 Criteria-Based Optimization

As already mentioned the criteria for determining a "good" partitioning depend heavily on the concrete application. For enabling adjustment to different application requirements, the parameters that influence the final partitioning can be adjusted to produce various resulting partitionings. A basic problem in this context is the fact that the relation between the parameters that can be modified to influence the result and the degree of fulfilment of the quality criteria is far from being evident. We addressed this problem by extending PATO to automatically respond to given criteria. Thus, the user does not need to understand the parameters used in the configuration of our tool. Instead, the user can specify relevant criteria for the application at hand. Based on this information, the system systematically applies different parameter configurations and selects the configuration that maximizes the quality of the resulting partitioning based on the parameters specified by the user.

Without this support for criteria-based optimization, a user would have to try different configurations, find parameters that influence the partition quality for his application and test how these parameters have to be adjusted to improve the partitioning. The first step towards automatical configuration is to try different configurations automatically and assess the resulting partitioning with respect to the needs of the application.

PATO automates this process by choosing a configuration P of parameters that maximizes the weighted sum of a set of criteria values $v_{c,p}$ using the following formula:

$$\max_{p \in Config} \sum_{c \in C} w_c \cdot v_{c,p}$$

Here C is a set of criteria and w_c is the weight given to a criterion $c \in C$. A basic problem associated with this approach is the determination of appropriate weights to be used in the formula above.

4.1 Weight Determination

While the partition criteria for an application can normally be determined by the user, it is more sophisticated to determine the relative importance of criteria and assign weights accordingly.

Some criteria directly correspond to expense factors of an application, for example, in distributed reasoning there is a trade off between communication cost (caused by separation) and benefits from parallelization. It is not that clear however how the expenses caused by an additional link between modules compares to the costs of increasing the size of the largest module by 2%. In many applications, criteria are only known to have positive or negative influence on the performance and the weight is not determined.

In this case, a solution is to make use of boolean criteria. A boolean criterion *bool* has values $v_{bool,p} \in \{0, -\infty\}$ corresponding to $\{true, false\}$. In particular, it makes sense to apply at most one non-boolean criterion (i.e. the user chooses one criterion that is maximized and specifies a range of acceptable values for each other criterion). Then the best configuration is one that complies with all boolean criteria and maximizes the non-boolean criterion.

In cases where a criterion is used that precisely measures the quality of a partitioning for a given application but is costly to compute (e.g. the runtime of an algorithm that uses the partitioning), it can be approximated by combining other criteria. Here the weights are determined by first testing a small sample of configurations and determining the best configuration based on the exact criterion. Then the weights we are looking for $(w_1, w_2, \ldots, w_n) = \overrightarrow{w}$ are vectors that solve the inequation $(E \cdot \overrightarrow{v} - V) \cdot \overrightarrow{w} \geq 0$ where $\overrightarrow{v} = (v_1, v_2, \ldots, v_m)$ are the criteria values of the best configuration, the matrix V consists of the criteria value vectors for each sample configuration and E is a $n \times m$ matrix of ones.

4.2 Dependency Visualization

When relative weights are not determined and acceptable criteria ranges are given instead, the result of criteria maximization is a set of acceptable configurations.

For choosing from these configurations PATO generates tables of the relevant criteria and parameters that can be read into Matlab or Scilab for visualization. Figure 1 gives an example of this visualization which is created by a single Scilab statement[2]. Furthermore, the pareto-optimal configurations can be determined using the Scilab or Matlab.

4.3 Advanced Search

Currently, PATO chooses the optimal configuration from a set of configurations that were performed before. This is the basis for implementing an optimization mechanism that chooses the configuration to determine the best configuration for a new problem based on the previous results. Starting with a default configuration, it is improved step by step according to the given criteria performing a greedy strategy: The neighboring configurations, i.e. the configurations with one parameter increased or decreased, are tried and the best of these is chosen, then the neighbors of the current best configuration are tested.

Performance of this optimization strategy can be improved by applying binary search instead of stepwise increasing/decreasing parameters. If increasing the value of parameter p from 4 to 5 improved the result, the maximal value $pmax$ for p is tried next, then the median of $pmax$ and 5 is tried. For criteria that do not decrease/increase monotonically when one parameter is changed, random configuration changes are performed to avoid getting stuck in local criteria maxima (random walk strategy).

5 Example Applications

We tested the criteria-based partitioning method described above in two example applications that have rather different reuirements with respect to the resulting modules. The first application example is concerned with supporting distributed reasoning about description logic ontologies. As an example ontology we used the *Aminoacid* ontology, a rather expressive description logic ontology from the biomedical domain [3]. This application requires a partitioning to reason efficiently on a knowledge base which basically requires minimizing communication costs between modules. The second example application is concerned with visualizing the content of a very large ontology in a survey style. As a basis for this application, we use the NCI ontology, a very large ontology from the medical domain. In this second application, the goal is to produce a limited number of modules that can be displayed in a single image for showing major topics and relations between modules. In the following, we describe the applications in more details and discuss the use of criteria-based partitioning introduced above.

[2] `surf(heightthreshold,maximumislandsize,edgecut,(sizebalance<=2).*(nrOfModules>=3));`

[3] http://www.co-ode.org/ontologies/amino-acid/2006/05/18/amino-acid.owl

5.1 Reasoning

Modularization is a way to support efficient reasoning about very large ontologies as it supports distributed and partially parallelized reasoning methods that promise to scale better than existing centralized reasoning approaches. If the ontology to be reasoned upon is not already provided in a modular way, partitioning is necessary before the reasoning can be performed. In previous work, we have developed a sound and complete distributed reasoning method for large ontologies specified in $\mathcal{ALC}$ - a significant subset of logic underlying the Web Ontology language - based on resolution [8].

In this reasoning method an ontology is first translated into first order clauses which are subsequently distributed based on a partitioning of the ontology terms. Based on this partitioning, every clause is mapped to specific module and inference is restricted to local reasoning. I.e., for a rule to be applicable the premises have to be present in the same module. In our experiments, we used the PATO System to determine a partitioning of the terms in the ontology that improves distributed reasoning by reducing the communication between modules necessary to achieve completeness. The requirements of the reasoning application are:

- Communication cost should be as low as possible.
- Computation load should be distributed uniformly among the modules.
- There should be a significant number of modules in order to profit from the distribution. In the experiments, we arbitrarily determine such a significant number to be at least 4.

The experiments were executed using the AMINOACID[4] ontology, a relatively rich $\mathcal{ALCF}$ ontology. Because our reasoning method can currently not handle functional properties we approximated the ontology by removing the two functional type statements.

In the reasoning methods, communication between two models is necessary if terms of the ontology that are placed in different modules occur in the same axiom. To model this, the ontology graph creation was set to include an edge between every pair of ontology terms that occur in the same axiom. Taking the module graph as a basis the requirements mentioned above can be modeled using the following built-in criteria:

- weight - edge cut = -1
- range - size balance = $[-, 2]$
- range - number of modules = $[3, -]$

[4] http://www.co-ode.org/ontologies/amino-acid/2006/05/18/amino-acid.owl

This means that we consider all partitioning where the size balance is smaller than 2 and the number of modules is larger than 3 and let PATO chose a parameter configuration that minimizes the edge cute. This setting is reasonable, because the edge cut approximates the communication cost and should be minimized, therefore the weight is set to a negative number.

The result is shown in Figure 1. The graph in this figure depicts the value of *edge cut* for the parameters *maximum island size* and *height threshold*. The bright surface color indicates that for the corresponding configurations both *balance* and *number of modules* are in the specified range. Hence, a good configuration is a bright and low point on the surface.

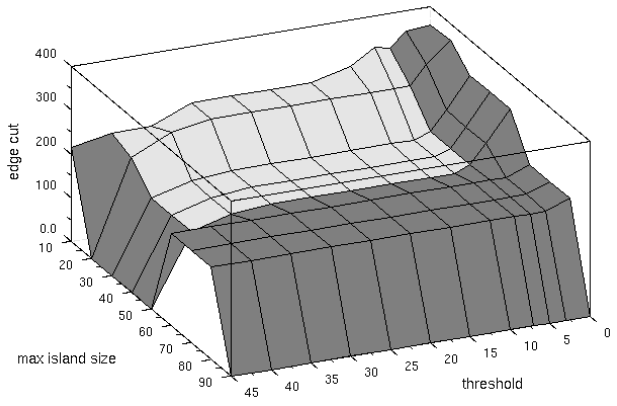

Figure 1. Evaluation graph for the distributed reasoning application depicting the value of the *edge cut* criterion.

In order to verify the hypothesis that *edge cut* is indeed a good indicator for our communication cost we implemented an application specific criterion using the corresponding interface described above. The new criterion is determine by wrapping the actual distributed reasoner, that is called with the current partitioning and returns the resulting communication effort. Figure 2 shows that the actual communication is not exactly proportional to the edge cut values, but the best configurations detected by the edge cut criterion are also the best ones determined by the actual communication (the corresponding communication value is 733). This means that it is reasonable to use the edge cut criterion for determining optimal parameter settings for partitioning ontologies as a basis for distributed reasoning.

For judging the quality of the partitioning created by PATO we compared it to a partitioning method that directly minimizes communication cost. The latter method first performs a test reasoning task on a completely distributed ontology (each ontology term is placed in a separate mod-

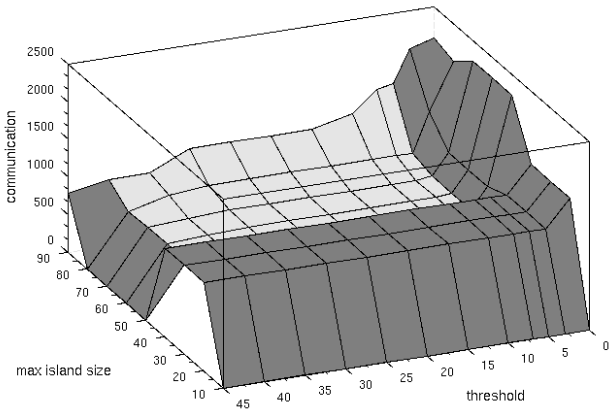

Figure 2. Evaluation graph for the distributed reasoning application depicting the value of the *communication* criterion.

number of modules	2	3	4	5
PATO				
derived	6686	6701	6701	9042
communication	703	733	782	1512
com./derived	11%	11%	12%	17%
balance	1.07	1.60	2.13	1.80
Com.-based				
derived	6173	7184	7156	8261
communication	650	1167	1293	1437
com./derived	11%	16%	18%	17%
balance	1.02	1.21	1.19	1.40

Table 1. Comparision of communication effort in the distributed reasoning application using PATO partitioning and communication-based partitioning.

ule) and records each axiom that is propagated from one module to another. Based on this reference communication the initial modules were merged to reduce communication using the graph algorithm provided by METIS[5]. We compared the communication results of the PATO partitioning on the same reasoning task the communication-based partitioning was optimized for.

The results depicted in Table 1 show that the PATO partitioning is comparable to communication-based partitioning. For 3 and 4 modules, the number of derivations and communicated clauses is smaller using the PATO partitioning, while for 2 and 5 modules communication-based partitioning performs better. Communication-based partitioning is not optimal, because in addition to minimizing the communication the algorithm forces the modules to perform about the same number of derivations. For measuring the distribution of computation load we use the *balance* value defined by the number of axioms derived in the most active module devided by the average number of derivations per module. Note that performing the test reasoning task is only possible in our simulation. In realistic distributed reasoning applications, the ontologies are much too large to be handled by a single reasoner (otherwise centralized reasoning would be more efficient). Hence, performing a test reasoning task is not feasible. As these experiments show, using PATO in such situations is reasonable as it comes close to the inaccessible optimal solution.

[5]http://glaros.dtc.umn.edu/gkhome/views/metis/

5.2 Visualization

To substantiate our claim that the PATO System can be adapted to applications with very different needs, we applied it in a second scenario that is substantially different from the distributed reasoning application described above. This second scenario is concerned with supporting the user in the exploration of a very large ontology by creating suitable visualizations of subparts of the ontology.

For producing a useful visualization of the whole ontology, the number of modules should be about 30 as this is a number of modules that can still be shown on a single display without overwhelming the user. Furthermore very large modules should be avoided. Therefore the criteria requirements are set to

- range - number of modules = $[28, 32]$
- range - size balance = $[-, 4]$

We demonstrate this using PATO on the NCI ontology a very large OWL Ontology containing about 26.000 concepts [4]. In order to make this ontology more accessible to the user, we used PATO to partition the ontology into smaller modules and visualized the resulting module graph using the Pajek[6], a tool for large network analysis. The network shown in figure 3 displays the module graph, the size of each vertex corresponding to the number of terms in the module. In addition to visualization, we used Pajek for determining the module labels. In particular, a module is labeled by the vertex with the highest *betweenness*[7], a central-

[6]http://vlado.fmf.uni-lj.si/pub/networks/pajek/

[7]The betweenness of a vertex $v \in V$ is the percentage of shortest paths this vertex lies on: $betweenness(v) = \sum_{\substack{s,t \in V \\ s \neq v \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$
were σ_{st} is the number of shortest paths between the vertices s and t, and $\sigma_{st}(v)$ is the number of shortest paths between s and t that v lies on.

ity measurement defined by [3] for social networks. Based on the criteria shown above, PATO chooses a configuration that results in the partitioning depicted in Figure 3.

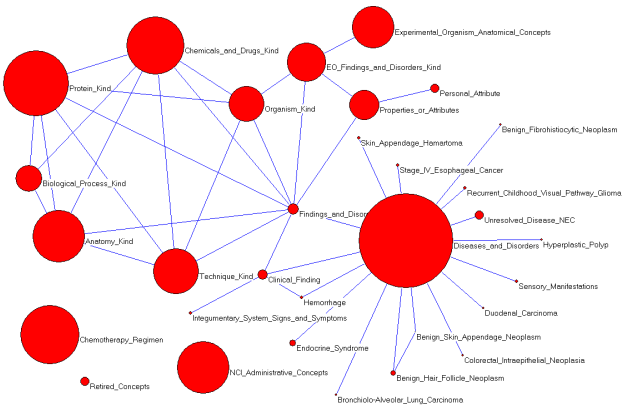

Figure 3. The module graphs displays the connections between modules. For each module the name of the vertex with the highest centrality value labels the module.

As the image shows, using PATO to partition a very large ontology like the NCI ontology and visualizing it with the Pajek system provides a good first impression on the contents of the ontology. For instance, we can see that the majority of the ontology is about different kinds of diseases. Further, we see that organisms, chemicals and drugs as well as anatomy and biological processes are also important topics covered in the ontology. Further, we can find out more about the different topics by loading the different modules created by PATO into an OWL editor for further inspection.

6 Conclusion

We presented a partitioning method for large ontologies that semi-automatically adapts to given requirements. In developing this method, we respond to the fact that different application scenarios come with different requirements concerning the desired nature of the modularization. The greatest challenge in providing a flexible approach to partitioning is to establish a link between the requirements of the application and the right choice of parameters for the partitioning method that promises to provide the best result. We addressed this problem by providing a semi-automated mechanism that determines the best parameter configuration based on a definition of a number of criteria defined over the resulting modular ontology. We implemented a number of generally useful criteria and provided the possibility of introducing user-defined criteria. While this is a big step towards a general purpose partitioning method, identifying the right criteria for a given situation is still a challenge and requires a deeper understanding of the application and graph theoretic notions currently used as criteria. An obvious limitation of the approach with respect to applications that require some form of reasoning with the knowledge encoded in the ontology that is partitioned is the fact, that the criteria used are purely syntactical. Other researchers have investigated logical criteria for partitioning ontologies that provide certain guarantees about the completeness of reasoning in the partitioned ontology. Introducing such logical criteria (e.g. the notion of conservative extensions for description logics) as Boolean Criteria for partitioning into the system could further improve the approach.

References

[1] V. Batagelj. Analysis of large networks - islands. Presented at Dagstuhl seminar 03361: Algorithmic Aspects of Large and Complex Networks, August/September 2003.

[2] B. Cuenca Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularity and web ontologies. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR2006)*, 2006.

[3] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[4] J. Golbeck, G. Fragoso, F. Hartel, J. Hendler, J. Oberthaler, and B. Parsia. The national cancer institute's thesaurus and ontology. *Journal of Web Semantics*, 1(1), 2003.

[5] B. MacCartney, S. McIlraith, E. Amir, and T. Uribe. Practical partition-based theorem proving for large knowledge bases. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

[6] A. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In *Proceedings of the 16th International FLAIRS Conference*. AAAI, 2003.

[7] A. Schlicht and H. Stuckenschmidt. Criteria-based partitioning of large ontologies. In *Proceedings of the International Conference on Knowledge Capture (K-CAP)*, 2007. Poster Contribution.

[8] A. Schlicht and H. Stuckenschmidt. Distributed resolution for $\mathcal{ALC}$ - first results. In *Workshop on Advancing Reasoning on the Web: Scalability and Commonsense ESWC*, 2008.

[9] H. Stuckenschmidt. Network Analysis as a Basis for Partitioning Class Hierarchies. In *Workshop on Semantic Network Analysis ISWC*, 2006.

[10] H. Stuckenschmidt and M. Klein. Structure-based partitioning of large concept hierarchies. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, pages 289–303, Hiroshima, Japan, nov 2004.

[11] H. Stuckenschmidt and M. R. Menken. Tool Support for Dependency-Based Partitioning of OWL Ontologies. Technical report, 2005.

This work was partially supported by the German Science Foundation in the Emmy-Noether Program under contract Stu 266/3-1.