7

Structure-Based Partitioning of Large Ontologies

Heiner Stuckenschmidt and Anne Schlicht

Universität Mannheim, Germany
{heiner, anne}@informatik.uni-mannheim.de

Summary. In this chapter we describe a method for structure-based ontology partitioning and its implementation that is practically applicable to very large ontologies. We show that a modularization based on structural properties of the ontology only already results in modules that intuitively make sense. The method was used for creating an overview graph for ontologies and for extracting key topics from an ontology that correspond to topics selected by human experts. Because the optimal modularization of an ontology greatly depends on the application it is used for, we implemented the partitioning algorithm in a way that allows for adaption to different requirements. Furthermore this adaption can be performed automatically by specifying requirements of the application.

7.1 Introduction

In our work, we focus on the task of splitting up an existing ontology into a set of modules according to some criteria that define the notion of a good modularization. Intuitively, we can say that a module should contain information about a coherent subtopic that can stand for itself. This requires that the concepts within a module are semantically connected to each other and do not have strong dependencies with information outside the module. These considerations imply the need for a notion of *dependency* between concepts that needs to be taken into account. There are many different ways in which concepts can be related explicitly or implicitly. At this point we abstract from specific kinds of dependencies and choose a general notion of dependency between concepts. The resulting model is the one of a weighted graph $O = \langle C, D, w \rangle$ where nodes C represent concepts and links D between concepts represent different kinds of dependencies that can be weighted according to the strength of the dependency. These dependencies can reflect the definitions of the ontology or can be implied by the intuitive understanding of concepts and background knowledge about the respective domain. Looking for an automatic partitioning method, we are only interested in such kinds of dependencies that can be derived from the ontology itself. This leads us to a first central assumption underlying our approach:

> **Assumption 1:** Dependencies between concepts can be derived from the structure of the ontology.

Depending on the representation language, different structures can be used as indicators of dependencies. These structures can be subclass relations between classes, other relations linked to classes by the range and domain restrictions or the appearance of a class name in the definition of another class. In previous work, we have shown that this assumption is valid in many cases [13]. A second basic assumption of our approach that directly follows from the first assumption and will be the focus of this paper is the following:

> **Assumption 2:** The quality of a modularization can be determined on the basis of the structure of the individual modules and the connections between them.

This assumption does not only provide a rationale for structure-based ontology partitioning, it also allows us to adapt the partitioning algorithm originally proposed in [13] by explicitly taking structural criteria for measuring the quality of the resulting modular ontology into account.

In the subsequent section we describe the different steps of the partitioning algorithm. Section 3 comprises an overview of the implementation including instruction for its utilization. We demonstrate application of the tool for visualization and identification of key topics in section 4. The last section summarizes the main ideas and results and gives an outlook on future work.

7.2 Algorithm

Our algorithm consists of three tasks that are executed in six independent steps. The first task (Steps 1.1 and 1.2) is the creation of a dependency graph from an ontology definition. Guided by this graph the actual partitioning is the second task. The third task is optimization of the partitioning by assignment of isolated concepts, merging some modules and duplicating selected axioms. Step 4 describes how parameters that are required by the different partitioning steps are determined automatically, based on a given set of criteria.

7.2.1 Dependency Graph

The first task of the algorithm is the conversion of an ontology in OWL, RDF or KIF format into a weighted graph. It consists of two steps, the creation of the graph and the computation of the weights.

Step 1.1: *Create Dependency Graph:* In the first step a dependency graph is extracted from an ontology source file. The elements of the ontology (concepts, relations, instances) are represented by nodes in the graph. Links are introduced between nodes if the corresponding elements are related in the ontology. There are five types of relations between elements to choose from for the creation of links: subclass, property, definition, substring and distance relations.

When property relations are to be included, domain and range of each property are connected by a link. Definition relations are established between a concept and terms contained in its definition (either only properties or also other resources). These can be used to make concepts dependent on some shared property. The remaining two relations, substring and string distance, look at the concept names (or labels if specified). They create a relation if one concept name is contained in another or if the string distance between two concept names is below a certain threshold. The string relations are oportune when the terms of the ontology have a

compositional structure. For example, [10] found that in the Gene ontology 65% of the terms encode a semantic relation in their name (e.g. *regulation of cell proliferation* is related to *cell proliferation*).

Step 1.2: *Determine Strength of Dependencies:* In the second step the strength of the dependencies between the concepts has to be determined. Following the basic assumption of our approach, we use the structure of the dependency graph to determine the weights of dependencies. In particular we use results from social network theory by computing the proportional strength network for the dependency graph. The strength of the dependency of a connection between a node c_i and c_j is determined to be the proportional strengths of the connection. The proportional strength describes the importance of a link from one node to the other based on the number of connections a node has. In general it is computed by dividing the sum of the weights of all connections between c_i and c_j by the sum of the weights of all connections c_i has to other nodes (compare [4], page 54ff):

$$w(c_i, c_j) = \frac{a_{ij} + a_{ji}}{\sum_k a_{ik} + a_{ki}}$$

Here a_{ij} is the weight preassigned to the link between c_i and c_j - in the experiments reported in this section this will always be one. As a consequence, the proportional strength used in the experiments is one divided by the number of nodes c_i is connected to.

The intuition behind it is that individual social contacts become more important if there are only few of them. In our setting, this measure is useful because we want to prevent that classes that are only related to a low number of other classes get separated from them. This would be against the intuition that classes in a module should be related.

We use node **d** in Fig. 7.1 to illustrate the calculation of weights using the proportional

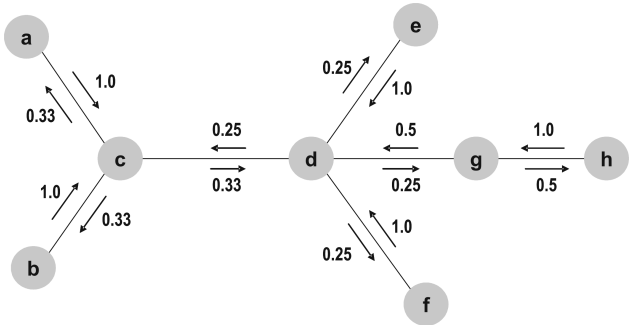

Fig. 7.1. An example graph with proportional strength dependencies.

strength. The node has four direct neighbors, this means that the proportional strength of the relation to these neighbors is 0.25 (one divided by four). Different levels of dependency between **d** and its neighbors now arise from the relative dependencies of the neighbors with **d** (the proportional strength is non-symmetric). We see that **e** and **f** having no other neighbors

completely depend on **d**. The corresponding value of the dependency is 1. Further, the strength of the dependency between **g** and **d** is 0.5, because **g** has two neighbors and the dependency between **b** and **d** is 0.33 as **b** has 3 neighbors.

7.2.2 Identification of Modules

Step 2: *Determine Modules* The proportional strength network provides us with a foundation for detecting sets of strongly related concepts. For this purpose, we make use of an algorithm that computes all maximal line islands of a given size in a graph [2].

Definition 1 (Line Island). *A set of vertices $I \subseteq C$ is a line island in a dependency graph $G = (C, D, w)$ if and only if*

- *I induces a connected subgraph of G*
- *There is a weighted graph $T = (V_T, E_T, w_T)$ such that:*
  - *T is embedded in G*
  - *T is an maximal spanning tree*[1] *with respect to I*
  - *the following equation holds:*

$$\max_{\{(v,v')\in D | (v\in I \wedge v'\notin I)\vee(v'\in I\wedge v\notin I)\}} w(v,v') < \min_{(u,u')\in E_T} w(u,u')$$

Note that for the determination of the maximal spanning tree the direction of edges is not considered. ◇

This criterion exactly coincides with our intuition about the nature of modules given in the introduction, because it determines sets of concepts that are stronger internally connected than to any other concept not in the set. The algorithm requires an upper and a lower bound on the size of the detected set as input and assigns an island number to each node in the dependency graph. We denote the island number assigned to a concept c as $\alpha(c)$. The assignment $\alpha(c) = 0$ means that c could not be assigned to an island.

We use different sets of nodes in the graph in Fig. 7.1 to illustrate the concept of a line island. Let us first consider the set $\{\mathbf{a}, ..., \mathbf{f}\}$. It forms a connected subgraph. The maximal spanning tree of this set consists of the edges $\mathbf{a} \xrightarrow{1.0} \mathbf{c}$, $\mathbf{b} \xrightarrow{1.0} \mathbf{c}$, $\mathbf{c} \xrightarrow{0.33} \mathbf{d}$, $\mathbf{e} \xrightarrow{1.0} \mathbf{d}$, and $\mathbf{f} \xrightarrow{1.0} \mathbf{d}$. We can see however, that this node set is not an island, because the minimal weight of an edge in the spanning tree is 0.33 and there is an incoming edge with strength 0.5 ($\mathbf{g} \xrightarrow{0.5} \mathbf{d}$). If we look at the remaining set of nodes $\{\mathbf{g}, \mathbf{h}\}$, we see that it fulfills the conditions of an island: it forms a connected subgraph, the maximal spanning tree consists of the edge $\mathbf{h} \xrightarrow{1.0} \mathbf{g}$ and the maximal value of in- or outgoing links is 0.5 ($\mathbf{g} \xrightarrow{0.5} \mathbf{d}$). This set, however, is not what we are looking for because it is not maximal: it is included in the set $\{\mathbf{d}, ..., \mathbf{h}\}$. This set is a line island with the maximal spanning tree consisting of the edges $\mathbf{e} \xrightarrow{1.0} \mathbf{d}$, $\mathbf{f} \xrightarrow{1.0} \mathbf{d}$, $\mathbf{g} \xrightarrow{0.5} \mathbf{d}$ and $\mathbf{h} \xrightarrow{1.0} \mathbf{g}$ where the minimal weight (0.5) is higher than the maximal weight of any external link which is $\mathbf{c} \xrightarrow{0.33} \mathbf{d}$. Another reason for preferring this island is that the remaining node set $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ also forms a line island with maximal spanning tree $\mathbf{a} \xrightarrow{1.0} \mathbf{c}$, $\mathbf{b} \xrightarrow{1.0} \mathbf{c}$ and the weaker external link $\mathbf{c} \xrightarrow{0.33} \mathbf{d}$.

[1] A maximal spanning tree is a spanning tree with weight greater than or equal to the weight of every other spanning tree.

The actual calculation of the islands is done by an external program written by Matjaz Zaversnik[2]. This program requires specification of minimum and maximum island sizes to compute the above defined line islands. The minimum size is always set to 1 for not restricting the island creation more than necessary.

7.2.3 Optimization 1: Assignment of Isolated Concepts

After partitioning, in some cases there will be some leftover nodes which are not assigned to any cluster. The algorithm will automatically assign these nodes based on the strength of the relations to nodes already assigned to a module.

Step 3.1: *Assign Isolated Concepts* Leftover nodes are assigned to the cluster to which they have the strongest connection. In particular this is the island of the neighboring node they have the strongest relation to. In cases where all neighboring nodes are unassigned as well, these nodes are assigned first.

How this process works can best be explained by an example. Fig. 7.2 shows an example network. It contains two modules (M1 and M2) and one leftover node (c8). c8 is connected to module M1 by one edge with strength 0.3 and to module M2 by two arcs with strengths 0.2 and 0.3. To determine the strength of a connection between a leftover node and a module, the

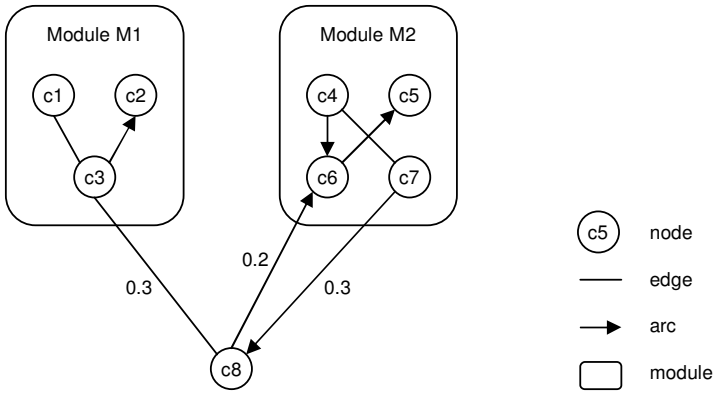

Fig. 7.2. Example network for the assignment of leftover nodes to modules.

strengths of all edges and arcs that connect the two are summed. Because edges are undirected and work in this respect in both directions, they can be considered twice as strong as arcs. Therefore their weight is doubled. In the example network the connection between c8 and M1 is 0.6, between c8 and M2 0.5, so the leftover node will be assigned to module M1.

7.2.4 Optimization 2: Merging

Looking at the result of the example application we get a first idea about the strengths and weaknesses of the algorithm. We can see that the algorithm generates some modules that meet

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

our intuition about the nature of a module quite well. In some cases subtrees that could be considered to form one module are further split even if the complete subtree does not exceed the upper size limit. This can be explained by an unbalanced modelling of the ontology as subtrees tend to be split up at concepts with a high number of direct subclasses compared to its sibling classes. This phenomenon often reflect a special importance of the respective concept in the ontology that also justifies the decision to create a separate model for this concept. The iterative strategy frees us from determining a lower bound for the size of modules. As a result, however, the algorithm sometimes create rather small modules. This normally happens when the root concept of a small subtree is linked to a concept that has many direct subclasses. For the result of the partitioning method these subsets are often pathological because a coherent topic is split up into a number of small modules that do not really constitute a sensible model on their own.

When inspecting the dependencies in the relevant parts of the hierarchy, we discovered that most of the problematic modules have very strong internal dependencies. In order to distinguish such cases, we need a measure for the strength of the internal dependency. The measure that we use is called the 'height' of an island. It uses the minimal spanning tree T used to identify the module: the overall strength of the internal dependency equals the strength of the weakest link in the spanning tree.

$$height(I) = \min_{(u,u') \in E_T} w(u, u')$$

We can again illustrate the the concept of height using the example from Fig. 7.1. We identified two islands, namely $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and $\{\mathbf{d}, ..., \mathbf{h}\}$. As the maximal spanning tree of the first island consists of the two edges $\mathbf{a} \xrightarrow{1.0} \mathbf{c}$, $\mathbf{b} \xrightarrow{1.0} \mathbf{c}$, the height of this island is 1.0. In the maximal spanning tree of the second island the edge $\mathbf{g} \xrightarrow{0.5} \mathbf{d}$ is the weakest link that therefore sets the height of the island to 0.5.

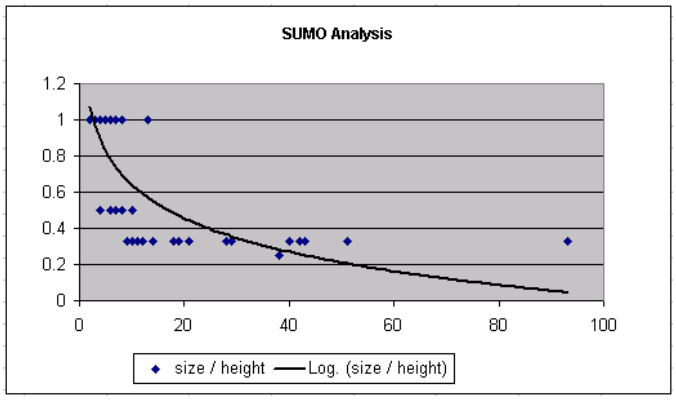

Fig. 7.3. Sizes and heights of partitions in the SUMO ontology

We found many cases where generated modules that do not make sense had an internal dependency of strength one. In a post-processing step this allows us to automatically detect critical modules. While for the case of an internal strength of one we almost never found the corresponding module useful in the context of the original ontology, it is not clear where to draw the line between a level of internal dependency that still defines sensible modules and a level that overrules important dependencies to concepts outside the module. In our experiments we made the experience that a threshold of 0.5 leads to good results in most cases.[3]

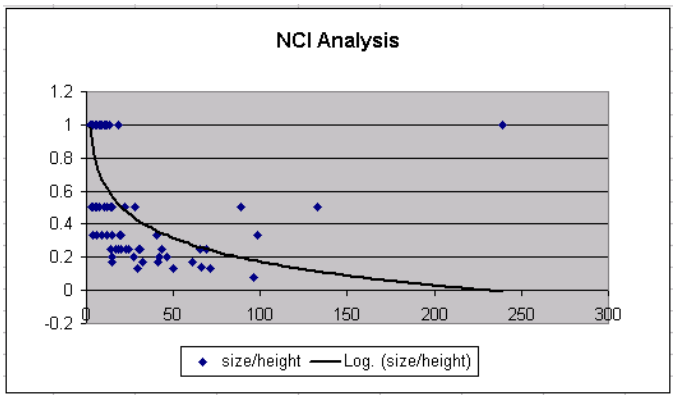

Fig. 7.4. Sizes and heights of partitions in the NCI ontology

Figures 7.3 and 7.4 show the results of comparing the size and the height of computed islands. The plots clearly show a correlation between these properties. We also see that—except for one case—islands with a height of one are quite small.[4] The results of these experiments provided us with sufficient evidence that the height of an island is a useful criterion for judging the quality of a module. As described above, one of the findings in the first experiments was the strong correlation between size of modules and the degree of internal dependency. Further, we found out that small modules were unnatural in most cases. In a second experiment, we show that this result can be used to 'repair' the result of the straightforward partitioning.

Step 3.2: *Merging* All modules whose height reaches the given threshold are merged into adjacent modules with a lower height. In many cases, there is only one adjacent module to merge with. In cases where more than one adjacent module exist, the strength of the dependencies between the modules is used to determine the candidate for merging.

[3] Note that due to the calculation of the dependency value, the internal strength is always of the form $\frac{1}{n}$.

[4] The exception is a part of the NCI ontology that lists all countries of the world and therefore contains 1 class with more than 200 subclasses

7.2.5 Optimization 3: Criteria Maximization

Starting point of a partitioning problem is usually an application in need for a partitioning that meets certain requirements. Investigation of applications for ontology modularization reveals that the criteria for determining a "good" partitioning depend heavily on the concrete application [6]. For enabling adjustment to different application requirements the parameters that influence the final partitioning are customizable.
In order to support users to chose the right setting for a given application, we do not force to directly provide values for the parameters mentioned above. Instead, the user is asked to select and rank quality criteria for the resulting partitioning. This frees the user from the need to understand the partitioning method and the influence of the different parameters. In contrast, the user only has to be concerned with the requirements of the application at hand. We believe that this is a big step towards enabling domain experts with limited knowledge about the technical details of representation languages for ontologies to use this technology.
It can be assumed that different applications may not only impose different weights for the built-in criteria but also require consideration of new criteria that are defined by the user. For usage of additional criteria our tool provides a simple interface. We assume that althought there may be various types of requirements, they all can be described by concrete measurements that map partitionings to decimal numbers. Different applications may share some measurements but disagree on their relative importance, it could even happen that one measurement is positive for one application and negative for another.

Criteria

The most obvious criteria used for optimization are *number of modules*, *avarage module size* and *variance of size*. Uniformity of the size distribution, is measured by the criteria *bulkyness* and *granularity*. The intention of bulkyness is to indicate that some modules are to large, e.g. the largest module has almost the same size as the whole ontology. For obtaining a smooth function module sizes $n_i = |M_i|$ are mapped to values in $[0, 1]$ depending on the size n of the ontology.

$$bulkyness(n_i, n) = \frac{1}{2} - \frac{1}{2}\, cos(\pi \cdot \frac{n_i}{n})$$

These values are averaged over all modules (weighted by module sizes) to obtain a measurement for the whole partitioning. Similarly, granularity indicates that modules are to small, e.g. when half of the concepts are contained in modules of size 1.

$$granularity(n_i, n) = (\frac{1}{2} + \frac{1}{2}\, cos(\pi \cdot \frac{n_i}{n}))^{20}$$

The precise value of the exponent does not matter, 20 is a value that worked well in practice. The graphs depicted in Fig. 7.2.5 illustrates the choice of functions. If there are two modules of the same size the bulkyness value is 0.5. If one of these modules is further partitioned into modules with size 1% of the size of the ontology, the granularity value is 0.5. In addition to criteria computed from size and number of modules, *connectedness* depends on the links between the modules. In many applications the number of symbols shared between axioms in different modules should be as small as possible. We consider the fraction of inter-modules edges with respect to the total number of edges. A detailed investigation of this measurement can be found in [12].

$$connectedness = \frac{\#\{(v, v') \in D \mid \alpha(v) \neq \alpha(v')\}}{\#\{(v, v') \in D\}}$$

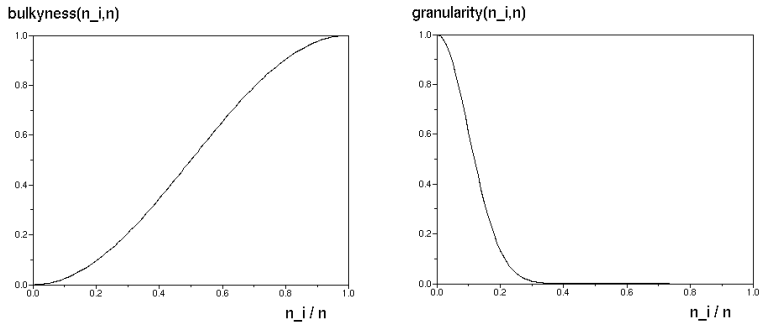

Fig. 7.5. bulkyness and granularity of modules against the relative size

where D is the set of edges of the dependency graph and α the assignment to modules (see Def. 1).
For optimization depending on specific terms contained in the ontology two additional criteria are defined. *Number of relevant modules* and *relative size of relevant modules* are computed depending on a given set of terms, i.e. a module is considered relevant if it contains one of the terms. For the relative size, the sum of the sizes of the relavant modules is compared to the size of the ontology.

There are different mechanisms for optimizing the configuration according to given criteria:

Dynamic defaults

Firstly, parameters that are not given in the settings file are determined automatically depending on given criteria. If for example the parameter "maximum island size" is not set in the input, it is set to $\frac{number\ of\ terms}{10} \cdot \frac{connectedness\ weight}{bulkyness\ weight}$. These dynamic default settings constitute an approximation of the optimal configuration.

Axiom Duplication

The preceding partitioning and optimization steps result in a non-redundant distributed representation of the source ontology. A term can not be allocated to more than one module. Sometimes it can be beneficial, however, to include certain axioms in several modules to decrease the connectedness of the resulting modularization. As on the other hand, copying axioms increases the redundancy, there has to be a tradeoff between these two conflicting requirements. Currently, this can be specified by setting an upper bound for the acceptable redundancy introduced. This means that the maximal redundancy is another parameter that influences the quality of the resulting partitioning in terms of connectedness and redundancy, Algorithm 2 shows the method for duplicating axioms based on a maximal redundancy value.[5]

[5] A partitioning is represented by a set of modules which in turn are represented by sets of axioms

Step 3.3: *Axiom Duplication* Axioms with a high number of links to another module are copied to that module if connectedness is decreased by this duplication. Duplication stops when the maximum redundancy is reached.

Algorithm 2 Axiom Duplication

```
Require: partitioning: Set<Set<Axiom>>
Require: maxRedundancy: double
  limit = ∞
  candidates = ∅
  while redundancy < maxRedundancy & limit > 0 do
    for all (axiom,module) ∈ partitioning do
      if numberOfLinks(axiom,module) > limit then
        candidates.add(axiom,module)
      end if
    end for
    for all (axiom,module) ∈ candidates do
      if duplicating axiom to module decreases connectedness then
        duplicate(axiom, module, partitioning)
      end if
    end for
    limit = limit - 1
  end while
```

Automatic configuration

The most advanced feature of the algorithm is the ability to automatically determine an optimal configuration of parameter settings:

Step 4: *Criteria-Based Optimization* Based on a set C of criteria and their weights w_c a configuration p is chosen that maximizes the weighted sum of the criteria values $v_{c,p}$.

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

Figure 7.6 demonstrates the selection of the configuration. The highest point of the surface corresponds to the best configuration for the given criteria.

7.3 Tool Support for Automatic Partitioning

The algorithm described in the last sections is implemented in the *Par*titioning *To*ol Pato, a Java application with two interfaces. Firstly, it performs the partitioning interactively through a graphical user interface. Secondly, the configuration can be specified in the settings file directly, providing control over additional parameters. A default configuration is computed, if only the source ontology file is specified. The tool is freely downloadable[6] and licensed under

[6] http://webrum.uni-mannheim.de/math/lski/Modularization/

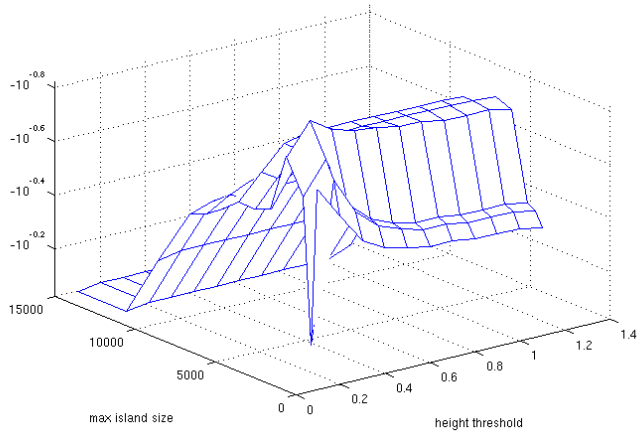

Fig. 7.6. Criteria-based determination of the configuration for the extraction application (see Sect. 7.4.3), displaying the value of $(-connectedness - 5 \cdot bulkyness)$

the GNU General Public License. The features that where added in Pato1.3 (mainly criteria-based optimization and OWL-output) are not yet supported by the GUI, they are controlled using the settings file.

7.3.1 Graph Generation

Figure 7.7 shows a screen shot of the tool in which an OWL ontology is converted to a dependency network. The screen is divided into three parts: the upper part gives a short help text about the currently selected tab, the middle part is for specifying the required arguments (in this case the input ontology and the output network) and the bottom part is for various optional parameters that influence the conversion. The tool converts an ontology written in RDFS or OWL to a dependency graph, written in Pajek format. Once the ontology is converted to a Pajek network, it can be visualized and further processed using Pajek, a network analysis program.[7].
This tool uses Sesame, a system for storing and querying data in RDF and RDFS.[3] The ontology is loaded into a local Sesame repository, after which it can be easily queried via an API. Because Sesame does not have native OWL support, some extra programming had to be done to deal with ontologies in this format. This includes explicitly querying for resources of type `owl:Class` while retrieving all classes (Sesame only returns resources of type `rdfs:Class`) and following blank nodes for determining the definition relations (see below).

To filter out irrelevant resources, the user can specify a number of namespaces that are to be ignored. They are entered in a text area (see Fig. 7.7). Resources that occur in those

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

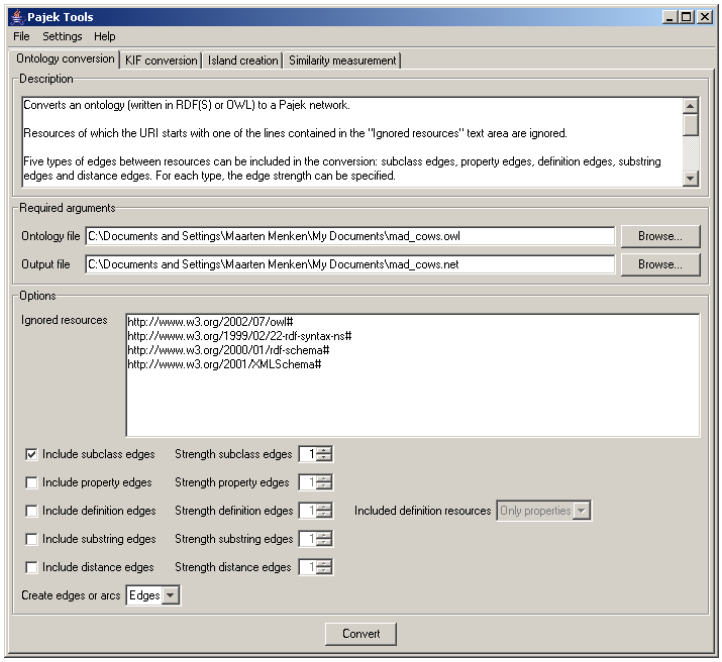

Fig. 7.7. Screen shot of the partitioning tool with the ontology conversion tab active.

namespaces do not show up in the resulting network. In most cases the classes and properties defined in RDFS and OWL can be ignored, and by entering the corresponding namespaces in the text area those resources are prevented from appearing in the network.

Before converting an ontology, the user has to decide what relations to include in the network, and if those relations are to be represented by edges (undirected) or arcs (directed). The tool allows five types of relations to be included: subclass, property, definition, substring, and string distance relations. The user also has to decide about the strength of each type. At the moment, only subclass relations that are explicitly stated in the ontology are included in the network. No reasoners are used to infer new subclass relations. This will be added in a future version.

A simple but effective feature added in Pato 1.3 is the option to select if the values of “rdfs:label” or “rdf:ID” are prefered for vertex labels. Now the use of “rdfs:label” can be turned of by setting “ontology conversion - use labels=false” for ontologies that use labels for verbose descriptions or other purposes.

7.3.2 Partition Generation and Improvement

The actual creation of the partitions is based on the previously generated dependency network. It iteratively splits the network into smaller parts until a specified maximum number of concepts per cluster is reached. Alternatively the occurence of very large and very small clusters is measured and the iteration stops at the weighted optimum of these measures.

The actual calculation of the islands is done by an external Windows program written by Matjaz Zaversnik[8]. On Unix Systems Pato searches for Wine[9] and tries to use this application for executing the Windows program.

7.3.3 Criteria-Based Optimization

The criteria-based optimization can be performed using build in analysis methods and/or additional criteria. Currently Pato computes connectedness of modules and some measures that depend on the size distribution of the modules. Either the partitioned graph structure and the final resulting distributed ontology are subject to analysis.
The relevant criteria are specified in the settings file, with weights indicating their relative importance. For example "criteria weight - connectedness=3.7" sets the importance of the connectedness-criterion.

It can be assumed that different applications may not only impose different weights for the build-in criteria but also require consideration of new criteria that are defined by the user. For usage of additional criteria the simple interface "Analyse" is provided that contains two methods, one for setting the input and one for getting the result, both represented as instances of java.util.Properties. The new analyse-class computes the value of one or more new criteria, its "getResult"-methode then returns the criteria name - criteria value pairs as an Properties-instance. The only thing to do apart from implementing the computation of the new criteria is declaring the name of the new class in the settings file (e.g. "analyse class=some.package.name. AnalyseImplementation"). After registration the new class is used automatically without recompiling Pato.

7.3.4 Visualization of Criteria Dependencies

If lists of parameters are specified for the parameters "maximum island size" and/or "height threshold", the corresponding criteria values are additionally stored as matrices. The produced file can be loaded into matlab or scilab for plotting the dependencies between criteria and parameters, Fig. 7.6 was created this way. Furthermore the matrices can be used for efficient determination of the pareto-optimal[10] configuration. Especially when relative importance of criteria is vague it might be necessary to try different weights, the optimal configurations for all possible weight assignments are the pareto-optimal configurations.

7.3.5 Module Graph Creation

For creating the ontologies overviews like displayed in Fig. 7.8 Pajek is used. Pato generates a network file (named "...net") for the graph and a corresponding vector file ("...vec") that defines the vertex sizes. The two files are loaded via Pajeks graphical user interface. Drawing is initiated by selecting "Draw-Vector" from the Draw-menu. For determination of vertex labels Pajeks centrality calculation is performed on the dependency graph created by Pato. The resulting vector file is in the settings file as value of "centrality vector file" prior to the module graph creation.

[8] http://vlado.fmf.uni-lj.si/pub/networks/
[9] http://www.winehq.org/
[10] A pareto-optimal configuration is a configuration that can not be improved for any criterion without degrading the value of another criterion.

7.3.6 Comparison and Evaluation

To evaluate the partitioning against some golden standard or against some other partitioning, the tool can calculate three similarity measurements: precision, recall and EdgeSim[9]. The first two measures are based on the numbers of intra-pairs, which are pairs of concepts that are in the same cluster[1]. The EdgeSim measure considers both the vertices and the edges and is not sensitive to the size and number of clusters (as are precision and recall). An intra-edge connects an intra-pair of vertices while an inter-edge connects vertices from different clusters.

Precision: The precision of a partitioning is defined as the ratio of intra-pairs in the generated partitioning that are also intra-pairs in the optimal partitioning.

Recall: The recall of a partitioning is defined by the ratio of intra-pairs in the optimal partitioning that are also intra-pair in the generated one.

EdgeSim: The EdgeSim measure is defined by the ratio of edges that are either intra-edges in both partitions or inter-edges in both partitions.

The three measures give an indication of how well the partitioning was performed and therefore what relations and strengths give best results.

7.4 Application

The main application area for Pato is visualization and identification of the key topics of an ontology. Visualization devides into the different tasks of visualizing a whole ontology by identifying modules and partitioning for visualization of single modules. Considering single modules is also relevant for facilitated reasoning and is related to module extraction.

7.4.1 Visualization of Large Ontologies

Module Graph

Apart from the resulting OWL-modules, Pato generates networks that can be visualized using Pajek[11], a tool for large network analysis. The network shown in Fig. 7.8 displays each module as a vertex, the size 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*[12], a centrality measurement defined by [8] for social networks.

For successful visualization of the whole ontology, the number of modules should be about 30 to provide as much information as can be displayed. Furthermore very large modules should be avoided. Therefore the criteria weights are set to (-1) for *connectedness* and (-2) for *abs(numberOfModules-30)*.

According to this criteria, Pato chooses the configuration[13]. Figure 7.9 shows the weighted sum of the criteria values. Dependency weights[14] where set directly, they depend on the type of relations that are to be visualized in the module graph.

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

[12] 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.

[13] height threshold=0.2, max island size=7000

[14] strength subclass links=7, strength property links=0.2, strength definition links=3

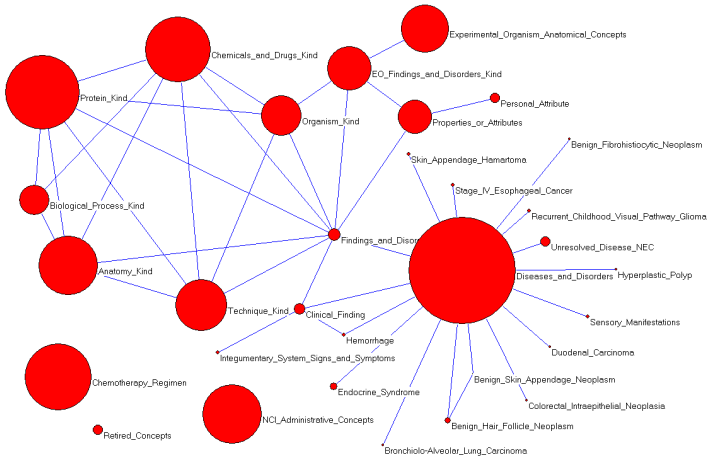

Fig. 7.8. The module graphs displays the connections between modules. For each module the name of the vertex with the highest centrality labels the module.

7.4.2 Identification of Key Topics

We consider an imaginary optimal partitioning of the ontology. An automatically generated partitioning is evaluated against this optimal partitioning in terms of recall and precision.
The basic problem of evaluating a partitioning is the fact, that in most cases we do not have an optimal partitioning to compare to. For these cases, we have to rely on alternative methods to determine the quality of the partitioning. A possibility that we will explore is empirical evaluation through user testing. Such an evaluation requires that the subjects have some knowledge about the domain modeled by the ontology. Therefore the ontology and the subjects have to be chosen carefully. The first option is to chose an ontology about a rather general topic (e.g. the transportation ontology). In this case any student is knowledgable enough to be chosen as a test subject. The other option is to chose a more specialized model and look for domain experts. Options here are the use of a computer science specific ontology (eg. the ACM classification) or a medical ontology. The advantage of the former is that test subjects are easier available while the time of medical experts is often rather limited.

A basic problem of empirical evaluation is the complexity of the task. Users will often not be able to oversee the complete ontology and to determine a good partitioning for themselves (in fact this is the reason why we need automatic partitioning). The most basic way of doing empirical evaluation is to directly use the notion of intra-pairs. As we have seen above, knowing all intra-pairs is sufficient for determining the quality measures defined above. This means that we can present pairs of concepts to subjects and ask them whether or not these concepts should be in the same part of the ontology. A problem of this approach is that the subject is not forced to be consistent. It might happen, that according to a subject A and B as well as A and C should be in the same part, but B and C should not. The second problem is the number of tests necessary to determine a partitioning. In the case of the ACM hierarchy, more that

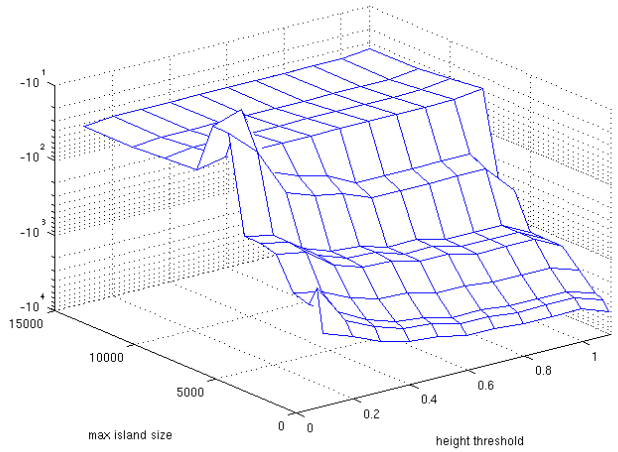

Fig. 7.9. The criteria evaluation for visualization displays the value of $-2 \cdot abs(numberOfModules - 30) - connectedness$.

1,5 Million pairs would have to be tested. In order to avoid these problems of consistency and scalability of empirical evaluation, we decided to perform an evaluation that is not based on concept pairs. The setting of our experiments is described in the following.

Setting

We used the ACM classification of computer science topics as a basis for performing an empirical evaluation of our partitioning method. The nature of the ACM hierarchy allows us to evaluate our method in terms of the number of key concepts identified when partitioning the model. The idea is that the root of each subtree distinguished by our partitioning algorithm should denote a unique subfield of computer science. When partitioning richer ontologies the hierarchy of one part would be a forest and centrality measures would be a better choice for denoting a subfield. However, for a partitioned hierarchy the subhierarchies are connected and the root is its superordinate concept. In order to determine such subfields that should be identified by the method, we analyzed the organization of computer science departments of Dutch universities with respect to the topics they used to identify subdivisions of their department. We then manually aligned these topics with the ACM topic hierarchy by translating the topic found into terms appearing in the ACM topic hierarchy. In cases where the topic matched more than one ACM terms (e.g. databases and information systems) both terms were counted. Terms that do not have a counterpart in the ACM hierarchy were ignored (e.g. 'mediamatics').

The test set consisted of 13 Dutch universities. Ten out of these had computer science departments. We extracted 85 terms from the corresponding web sites, mostly names of departments,

groups or institutes. We were able to map 77 of these terms into 42 distinct terms from the ACM hierarchy. We distinguish three subsets of these 42 terms: terms that occur at least once, terms that occur at least twice and terms that occur at least three times. We can assume that terms that occur more than once to be important subfields of computer science that we would like to capture in a single module.

Results

We compared these extracted terms with the root concepts of subtrees of the ACM hierarchy generated using our partitioning method. We chose to use a setting where the maximal size of an island is set to 100 and the threshold for merging islands is 0.2. With these settings, the method generated 23 modules. We decided to ignore three of the root terms:

ACM CS Classification This is the root of the hierarchy and not a proper term denoting a computer science topic

Mathematics of Computation The subtopics of this will normally be found in mathematics rather than computer science departments and were therefore not covered by our test set.

Hardware The subtopics of this module will normally be found in electrical engineering rather than computer science departments.

After this normalization, we compared the root terms of the generated modules given in Table 7.4.2 with the terms identified on the department web pages and used overlap to compute the quality of the partitioning in terms of precision and recall of our method.
From the web pages of Dutch computer science departments, we extracted the 42 ACM terms shown in Table 7.2. The most often occurring term was 'Algorithms' that described 5 groups, followed by 'Software' and 'Software Engineering'. Other frequently appearing topics were 'Robotics, 'Computer Systems', 'Computer Graphics', Information Systems', 'Expert Systems and Applications' (often referred to as 'Intelligent Systems'), Life Science applications, 'Systems Theory' and 'Theory of Computation'.

We can see that there is quite some overlap between the root nodes of the subtrees determined by our methods and the terms from the test set. The overlap is especially striking when we only consider the set of terms that occurred more than two times in the description of groups. Six out of these eleven terms where also determined by our method. The recall becomes worse when considering terms than only occurred twice or once. This was expected, however, because there are single research groups on more specific topics such as distributed databases that are not necessarily regarded as important subfields by a large majority of people. We included these terms with less support in the test set to evaluate how many of the terms found by our method are used to describe the topics of groups. It turns out that 12 out of the 20 terms occur in the test set leading to a maximal precision of 60% for the largest test set. We used to F-Measure $((2 * (precision * recall))/(precision + recall))$ to determine the overall quality of the results. It turns out that we receive the best results on the set of terms that occur at least twice. A summary of the results is shown in Table 7.3.
The main observation is that there is a significant overlap between topics that occur in the name of computer science research groups and the root nodes of the subtrees determined by our method. We were able to reach a precision of up to 60 percent when considering all terms occurring on the web sites. When only considering terms that are used more than two times, our method reached a recall of almost 55 percent. This can be considered a very good result as the chance of picking the most frequently occurring terms from the ACM hierarchy

1. Numerical Analysis
2. Image Processing and Computer Vision
3. Management of Computing and Information Systems
4. Computing Milieux
5. Software Engineering
6. Computer Communication Networks
7. Data
8. Information Storage and Retrieval
9. Operating Systems
10. Database Management
11. Computer Systems Organization
12. Information Interfaces and Presentation
13. Software
14. (Mathematics of Computing)
15. Theory of Computation
16. (ACM CS Classification)
17. Information Systems
18. Computer Applications
19. Simulation and Modeling
20. Artificial Intelligence
21. Computer Graphics
22. Computing Methodologies
23. (Hardware)

Table 7.1. The method determined 20 terms to represent important subareas of computer science. (Apart from the three nodes Mathematics of Computing, ACM CS Classification and Hardware)

is $\binom{11}{1300}$ (the binomial of 11 over 1300) and we do not have more information than the pure structure of the concept hierarchy.

This result supports our claim, that the structure of concept hierarchies contains important information about key concepts that in turn can be used to partition the hierarchy. Our hypothesis is, that this phenomenon is not random, but that people, when creating classification hierarchies are more careful when determining the subclasses of important classes. The result is a high number of children that cause our method to split the hierarchy at this particular point.

7.4.3 Visualization and Reasoning via Module Extraction

Large ontologies often cause problems for reasoning and editing. Furthermore the time needed for a human to overlook an ontology dramatically increases with its size. If not the whole ontology but only parts of it are relevant for an application the straight forward approach to dealing with too large ontologies is to consider only a part of it.

occurrence	ACM term
> 2	Algorithms
	Software
	Software Engineering
	Robotics
	Computer Systems Organization
	Computer Graphics
	Information Systems
	Applications And Expert Systems
	Life And Medical Sciences
	Systems Theory
	Theory Of Computation
> 1	User Interfaces
	Programming Techniques
	Artificial Augmented And Virtual Realities
	Artificial Intelligence
	Image Processing And Computer Vision
	Input/Output And Data Communications
	Parallelism And Concurrency
	Probability And Statistics
> 0	Computer-Communication Networks
	Business
	Computing Methodologies
	Control Design
	Decision Support
	Distributed Artificial Intelligence
	Distributed Databases
	Formal Methods
	Games
	Information Search And Retrieval
	Information Theory
	Management Of Computing And Information Systems
	Microcomputers
	Natural Language Processing
	Neural Nets
	Numerical Analysis
	Physical Sciences And Engineering
	Real-Time And Embedded Systems
	Security
	Signal Processing
	Software Development
	System Architectures
	Systems Analysis And Design

Table 7.2. ACM terms extracted from web sites of Dutch Computer Science Departments

Test Set	Precision		Recall		F-Measure
> 2	30%	6 of 20	54.55%	6 of 11	38.71%
> 1	40%	8 of 20	42.11%	8 of 19	41.03%
> 0	60%	12 of 20	28.57%	12 of 42	38.71%

Table 7.3. Summary of evaluation results

[5] describes a scenario in which knowledge is selected from online available ontologies. This knowledge selection procedure was applied to the semantic web browser plugin Magpie [7]. Magpie requires to automatically select and combine online available ontologies for identifying and highlighting instances of concepts in associated colors. Figure 7.10 depicts the sequence of tasks that have to be performed a detailed desciption of the application can be found in Chap. 3.1.

The partitioning method adressed in this chapter was applied for step 2, the extraction

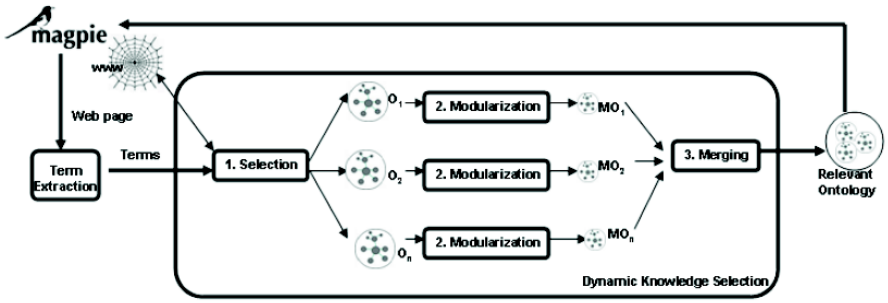

Fig. 7.10. The knowledge selection process and its use for semantic browsing with Magpie. Illustration from [6].

of relevant modules from the previously selected ontologies. A comparision of different extraction methods for this scenario is reported in [6], here we demonstrate how Pato is applied for the extraction process.

Usually knowledge extraction tools rely on a traversal approach and gather information from an ontology starting from a set of relevant terms. Nevertheless, in cases where knowledge is extracted repeatedly from the same ontology using a partitioning tool may be more efficient. The partitioning can be computed offline, repeated traversal of the whole ontology is replaced by the less complex selection of modules.

Setting

The scenario described above was simulated by manually extracting relevant keywords in news stories, using ontology selection tools[15]. In an example first described in [11] the keywords *Student*, *Researcher*, and *University* where used to select ontologies. Three ontologies covering these terms where obtained:

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

The appropriateness of Patos partitionings for this application is evaluated by two new criteria. First, the size of the obtained modules should be small with respect to the original ontology. Second partitioning with all relevant terms in one module are prefered i.e. the number of modules containing relevant terms should be small.

$$relativeSize\text{=-}40$$
$$numberOfRelevantModules^3\text{=-}1$$

The former criterion is the relative size of the resulting module compared to the size of the ontology, the latter criterion was emphazised by an exponent. In this application the relative importance of the two criteria can not be modelled by linear weights only because its gradient is not zero. By setting the exponent to 3 we make sure the rating of a partitioning is more affected by the decreasing the number of relevant modules from e.g. 3 to 2 than by the decreasement from 2 to 1.

Results

		ISWC	KA	PORTAL
configuration	max island size	40	10	15
	height threshold	1.1	1.1	0.2
	strength definition links	1	0	0
evaluation	relevant modules	1	2	1
	relative Size	0.54	0.14	0.27

Table 7.4. Configuration and evaluation for the knowledge extration setting.

Table 7.4.3 shows that for ISWC and KA the merging optimization was not necessary (the height ranges between 0 and 1, so merging modules with heigth 1.1 and larger means not

[15] in particular Swoogle (http://swoogle.umbc.ed)

merging at all). Due to the application requirement of small module size leftover nodes and small modules are not merged into larger modules. The low height threshold for PORTAL is caused by the second criterion. For forcing all terms into one module more merging was necessary.

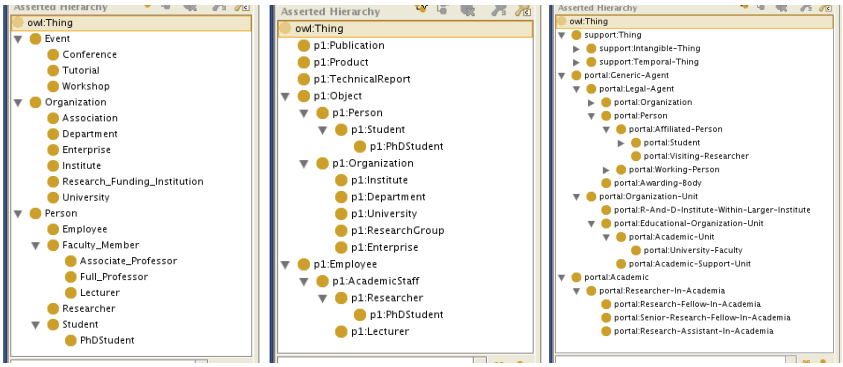

Fig. 7.11. Relevant modules of ISWC, KA and PORTAL ontologies for the terms researcher, student, university.

7.5 Conclusion

In this chapter we have described a method for structure-based ontology partitioning that is practically applicable to very large ontologies. The main idea of the algorithm is translating the structure of the ontology to a weighted graph. This graph is split up such that the resulting modules are stronger internally connected than externally connected. Finally three optimization steps are performed to improve the partitioning. Experiments on different ontologies have shown that a modularization based only on structural properties of the ontology already results in modules that intuitively make sense. Helpful visualization of large ontologies and extraction of key topics provided evidence for the appropriateness of the proposed approach.

Because modularizing an ontology essentially is a modeling activity, there is no “golden standard” to compare our results with. Actually, the notion of a “good” partitioning depends to a large extent on the application that uses the partitioned ontology. For enabling adaption to different application requirements we designed a parameterized partitioning algorithm. The parameters that determine the resulting partitioning are set automatically depending on given requirements. Encoding parameter selection as an optimization problem with respect to relevant quality criteria has been crucial for implementing automatic parameter selection. This disburdens the user from thinking about technical details of the algorithm and draws his attention to the requirements of the application at hand.

In future work we are planning to use our partitioning tool for distributed reasoning. The performance of distributed reasoning algorithms depends on additional properties like the size of the shared language. These requirements will be evaluated and implemented in the further development process of Pato. Thus it will be possible to configure the partitioning process to result in a reasonable trade-off between reasoning related requirements and maintainability requirements.

References

1. Nicolas Anquetil, Cédric Fourrier, and Timothy C. Lethbridge. Experiments with hierarchical clustering algorithms as software remodularization methods. In *Proceedings of the Working Conference on Reverse Engineering (WCRE'99)*, pages 304–313, Atlanta, USA, oct 1999.
2. Vladimir Batagelj. Analysis of large networks - islands. Presented at Dagstuhl seminar 03361: Algorithmic Aspects of Large and Complex Networks, August/September 2003.
3. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In Ian Horrocks and James Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, pages 54–68, Sardinia, Italy, jun 2002.
4. Ronald S. Burt. *Structural Holes. The Social Structure of Competition.* Harvard University Press, 1992.
5. Mathieu d'Aquin, Marta Sabou, and Enrico Motta. Modularization: a key for the dynamic selection of relevant knowledge components. In *Workshop on Modular Ontologies (WoMO)*, 2006.
6. Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt, and Martha Sabou. Ontology Modularization for Knowledge Selection: Experiments and Evaluations. In *International Conference on Database and Expert Systems Applications (DEXA)*, 2007.
7. Martin Dzbor, John Domingue, and Enrico Motta. Magpie - towards a semantic web browser. In *International Semantic Web Conference (ISWC)*, 2003.
8. Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.
9. Brian S. Mitchell and Spiros Mancoridis. Comparing the decompositions produced by software clustering algorithms using similarity measurements. In *Proceedings of the 17th IEEE International Conference on Software Maintenance (ICSM 2001)*, pages 744–753, Florence, Italy, nov 2001.
10. Philipp V. Ogren, Kevin B. Cohen, George Acquaah-Mensah, Jens Eberlein, and Lawrence Hunter. The compositional structure of gene ontology terms. In *Pacific Symposium on Biocomputing*, 2004.
11. Marta Sabou, Vanessa Lopez, and Enrico Motta. Ontology selection on the real semantic web: How to cover the queens birthday dinner? In *European Knowledge Acquisition Workshop (EKAW)*, Podebrady, Czech Republic, 2006.
12. Anne Schlicht and Heiner Stuckenschmidt. Towards Structural Criteria for Ontology Modularization. In *Workshop on Modular Ontologies ISWC*, 2006.
13. Heiner Stuckenschmidt and Michel Klein. Structure-based partitioning of large concept hierarchies. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, pages 289–303, Hiroshima, Japan, nov 2004.