# Network Analysis as a Basis for Partitioning Class Hierarchies

Heiner Stuckenschmidt

Vrije Universiteit Amsterdam
de Boelelaan 1081a, 1081HV Amsterdam
heiner@cs.vu.nl

**Abstract.** We discuss the use of network analysis methods to support the automatic partitioning of large concept hierarchies. Different from other work in the area, we directly apply these methods on the structure of the hierarchy. We show that this way of using network analysis techniques can provide significant results with respect to identifying key concepts and using them to determine subsets of class hierarchies that are related content-wise. We discuss the methods used and evaluate the result on the ACM classification of computer science topis.

## 1 Motivation

Network analysis techniques are recently receiving some attention in the area of semantic web research. People have recognized that information about communication patterns and social relationships can be used to better understand and design system behavior and to provide more efficient information sharing. Most existing work applies network analysis techniques in a rather standard way by analyzing relations between people (often users or their representation). Some work tries to enrich social networks with an analysis of shared topics or shared terminology [5]. The work reported in this paper takes a different approach. Rather than analyzing people, we use network analysis methods to analyze ontologies. For this purpose, we treat ontologies as networks where named elements in the ontology (concepts, relations, instances) are nodes. Links between these named elements are derived from the definitions and axioms in the ontology. Similar ideas are reported in [4].

Based on this network representation of an ontology, available analysis techniques provide us with a wide range of possibilities such as

- determining important concepts
- valuating the degree of dependency between two concepts
- finding sets of related concepts
- determining completely unrelated parts of an ontology
- etc.

In this paper, we focus on the use of network analysis for a particular task: the partitioning of an ontology into a number of disjoint and covering set of concepts. There are a number of use cases for this kind partitioning including distributed maintenance,

selective reuse [6] and efficient reasoning [1]. We have developed a method for partitioning large taxonomies based on the structure of the hierarchy [7]. We describe this method and extend the work reported in [7] in two ways:

- We propose a heuristics for improving the result of the partitioning method by merging strongly related subparts
- We present results from an empirical evaluation of the methods on the ACM classification of computer science topics

The paper is structured as follows. In section 2 we provide a brief overview of the partitioning method in terms of several steps that lead from a given concept hierarchy to an assignment of concepts to modules. In section 3 we discuss the network analysis methods used in our approach in more detail and give examples for their application. Section 4 reports the setting and results from an evaluation of the partitioning method. We conclude with a discussion of the method and the general idea of using network analysis methods for analyzing ontologies.

## 2 Overview of the Partitioning Method

In [7] we presented a method for automatically partitioning lightweight ontologies. In particular, the method was aimed at models that only consist of a concept hierarchy. We showed that using simple heuristics, we can create meaningful partitions of class hierarchies for the purpose of supporting browsing and visualization of large hierarchies. We briefly recapitulate the different steps of our method as the following discussions will be based on this information.

**Step 1: Create Dependency Graph:** In the first step a dependency graph is extracted from an ontology source file. The idea is that 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, e.g. because they appear in the same definition.

**Step 2: Determine strength of Dependencies:** In the second step the strength of the dependencies between the concepts has to be determined. This actually consists of two parts: First of all, we can use algorithms from network analysis to compute degrees of relatedness between concepts based on the structure of the graph. Second, we can use weights to determine the importance of different types of dependencies, e.g. we can decide subclass relations have a higher impact than domain relations[1].

**Step 3: Determine Modules** The proportional strength network provides us with a foundation for detecting sets of strongly related concepts. This is done using a graph algorithm that detects minimal cuts in the network and uses them to split the

[1] In the following we are dealing with the subclass relation as the only type on dependency and therefore ignore the concept of weights

overall graph in sets of nodes that are less strongly connected to nodes outside the set than to nodes inside.

**Step 4/5: Improving the Partitioning** In the last steps the created partitioning is optimized. In these steps nodes leftover nodes from the previous steps are assigned to the module they have the strongest connection to. Further, we merge smaller modules into larger ones to get a less scattered partitioning. Candidates for this merging process are determined using a measure of coherence.

## 3 Using Network Analysis Methods

The main rational for translating ontologies into a graph structure is the availability of sophisticated methods for analyzing graph structures. Experiments with different available methods for determining the strength of relationships between nodes in a graph as well as different methods for partitioning graphs into disjoint sets of nodes revealed that the use of the line island method [2] on a relative strength network provides useful results. In the following, we present the network analysis methods used in our approach, results of applying them on an existing ontology about public and private transportation are shown in the next section.

Essentially, network analysis methods are used for determining the strength of relationships between nodes in the graph (step 2), for identifying potential partitions (step 3) and for improving the partitioning by determining parts to be merged (step 5).

### 3.1 Determining Strength of Dependencies: Relative Strength

After the dependency graph for an ontology has been created in the way descried in the last section the strengths of the dependencies between the concepts have 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 [3], page 54ff):

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

Here $w_{ij}$ is the weight preassigned to the link between $c_i$ and $c_j$ - in the experiments reported below 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 Figure 1 to illustrate the calculation of weights using the proportional

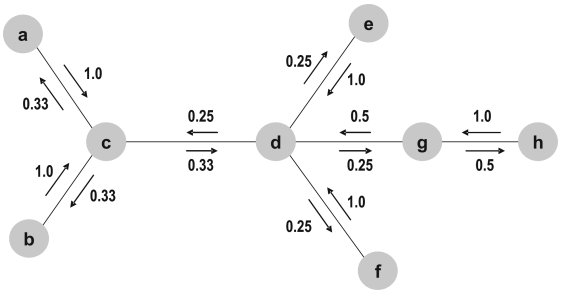


**Fig. 1.** An Example Graph with proportional strength dependencies

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.

### 3.2 Determine Modules: Line Islands

The proportional strength network provides us with a foundation for detecting sets of strongly related concepts that are candidates for forming a part in the partition. 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 $G = (C, D, w)$ - where $C$ is a set of nodes, $D$ a set of edged representing dependencies between them and $w$ is a set of weights of the edges - 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 with respect to $I$*
  - *the following equation holds:*

$$\max_{\{(v_i,v_j)\in D|(v_i\in I\wedge v_j\notin I)\vee(v_j\in I\wedge v_i\notin I)\}} w_{ij} < \min_{(v_k,v_l)\in E_T} w_{kl})$$

*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 Figure 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 $\mathbf{c} \xrightarrow{0.33} \mathbf{d}$.

### 3.3 Improving Partitions: Height of Line Islands

A problem of the use of the line island method for determining partitions is that it often creates a number of very small parts that only consists of two or three concepts. 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.

$$height(P) = \min_{(v_i,v_j)\in E_T} w_{ij}$$

We can again illustrate the the concept of height using the example from figure 1. We identifies 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 $\mathbf{g} \overset{0.5}{\rightarrow} \mathbf{d}$ is the weakest link that therefore sets the height of the island to 0.5.

The height of the island provides us with a criterion for automatically selecting parts of the graph to be merged again. In a series of experiments, we observed that most islands that with a height of 0.5 or more do not not correspond to meaningful modules and are therefore candidates for merging with other islands. A second choice that has to be made is about which other island to merge with. There are two factors that influence this decision. The first is the height of the other module. Here we prefer to merge with other islands that have a high height as well. The second factor is the strength of the relation between the two islands. We determine this strength by adding all edges that exist between nodes in the two islands. Based on these two factors, we determine the merging potential $m_{P_1}(P_2)$ of islands $P_1$ with island $P_2$ as follows:

$$m_{P_1}(P_2) = height(P_2) \cdot \sum_{v_i \in P_1, v_j \in P_2} w_{ij} + w_{ji}$$

Candidates for merging are now determined by ordering al islands based on their height. For each island with a height of 0.5 or more, we compute the merging potential with respect to all other islands. The island is merged with the one that has the highest merging potential. If the potential for different islands is the same, we chose the one with the highest height.

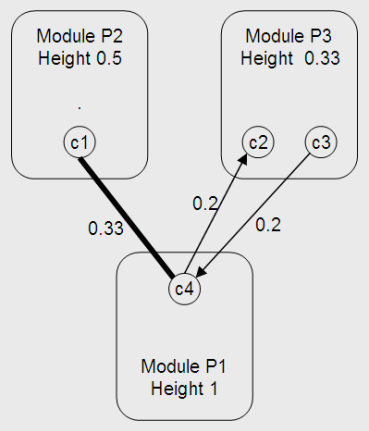


**Fig. 2.** Example of a merging decision

Figure 2 shows a simple example consisting of three islands (modules). Ordering the modules based on their height we get the sequence $P_1, P_2, P_3$. We start with $P_1$

which has the highest height and compute the merging potential with respect to $P_2$ and $P_3$ as follows:

$$m_{P_1}(P_2) = 0.5 \cdot 0.33 = 0.165$$
$$m_{P_1}(P_3) = 0.33 \cdot (0.2 + 0.2) = 0.132$$

As the merging potential with respect to $P_2$ is higher that the one with respect to $P_3$, we merge the two islands. As a result, the height of this new set of nodes becomes 0.33, because the edge between the two islands is now the one with the lowest weight. This means that we end up with two modules of height 0.33. No more merging operations are required.

## 4 Empirical Evaluation

We consider an imaginary optimal partitioning of the ontology. An automatically generated partitioning is evaluated against this optimal partitioning. The basis for comparison are pairs of classes. In particular, we consider pairs of concepts that are in the same part of the model These pairs are called intra-pairs, respectively. Based on the notion of intra-pairs, we can define two quality measures for a generated partitioning.

**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 in the generated one.

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 if 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 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.

### 4.1 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. 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.

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 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.

### 4.2 Results

We ran our method on the ACM classification using the hierarchy as the dependency graph. We further set the upper limit for the size of an island to 100 and the threshold value for merging islands to 0.2. The result are 23 subtrees with the following root nodes:

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)

As mentioned above, we disregarded the three nodes Mathematics of Computing, ACM CS Classification and Hardware. As a result, we receive 20 terms that our method determined to represent important subareas of computer science.

From the web pages of Dutch computer science departments, we extracted the 42 ACM terms shown in table 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'.

| 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 1.** Summary of evaluation results

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 1.

## 5 Discussion

We presented a method for automatically partitioning concept hierarchies using methods from social network analysis. We discussed the use of such methods for determining the strength of dependencies between classes, for determining sets of strongly related concepts and as a basis for a new heuristic for improving the result of the partitioning process. Further, we evaluate the method on the ACM classification of computer science topics by comparing the partitioning result with information about important computer science topics extracted from the web sites of (all) Dutch Computer Science Departments.

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 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

| occ. | 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 2.** ACM terms extracted from web sites of Dutch Computer Science Departments

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.

Of course the test set we used in our experiment is biased towards the particular strengthes of the Dutch Computer Science Community and does not necessarily reflect a neutral view on the importance of topics. We will try to overcome this problem in future work by repeating the experiment with computer science departments of a different country. Further, we plan to use human subject testing to support the current evaluation. We are currently preparing an experiment where people are asked to pick terms from the list of classes in the ACM hierarchy that they consider to represent important subfields of computer science. This test will be carried out in the computer science department of the Vrije Universiteit. As the members of the department have quite a number of different nationalities we hope to reduce the bias towards a particular nationality.

## References


1. E. Amir and S. McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 2005. Accepted for Publication.
2. V. Batagelj. Analysis of large networks - islands. Presented at Dagstuhl seminar 03361: Algorithmic Aspects of Large and Complex Networks, August/September 2003.
3. R.S. Burt. *Structural Holes. The Social Structure of Competition.* Harvard University Press, 1992.
4. Y. Kalfoglou. Using ontologies to support and critique decisions. In *Proceedings of the 1st International Conference on Knowledge Engineering and Decision Support (ICKEDS'04)*, Oporto, Portugal, July 2004.
5. Peter Mika. Flink: Using semantic web technology for the presentation and analysis of online social networks. *Journal of Web Semantics*, 2005. to appear.
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. 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.