Semantic Completeness in Sub-ontology Extraction Using Distributed Methods

Mehul Bhatt[1], Carlo Wouters[1], Andrew Flahive[1], Wenny Rahayu[1], and David Taniar[2]

[1] La Trobe University, Australia
{mbhatt,apflahiv,cewouter,wenny}@cs.latrobe.edu.au
[2] Monash University, Australia
David.Taniar@infotech.monash.edu.au

Abstract. The use of ontologies lies at the very heart of the newly emerging era of Semantic Web. They provide a shared conceptualization of some domain that may be communicated between people and application systems. A common problem with web ontologies is that they tend to grow large in scale and complexity as a result of ever increasing information requirements. The resulting ontologies are too large to be used in their entirety by one application. Our previous work, ***M****aterialized* ***O****ntology* ***V****iew* ***E****xtractor* (MOVE), has addressed this problem by proposing a distributed architecture for the extraction/optimization of a sub-ontology from a large scale base ontology. The extraction process consists of a number of independent optimization schemes that cover various aspects of the optimization process. In this paper, we extend MOVE with a Semantic Completeness Optimization Scheme (SCOS), which addresses the issue of the semantic correctness of the resulting sub-ontology. Moreover, we utilize distributed methods to implement SCOS in a cluster environment. Here, a distributed memory architecture serves two purposes: (a). Facilitates the utilization of a cluster environment typical in business organizations, which is in line with our envisaged application of the proposed system and (b). Enhances the performance of the computationally extensive extraction process when dealing with massively sized realistic ontologies.

Keywords: Parallel & Distributed Systems, Semantic Web, Ontologies, Sub-Ontology Extraction.

1 Introduction

The next generation of the internet is called the *semantic web*, and provides an environment that allows more intelligent knowledge management and data mining. The main focus is the increase in formal structures used on the internet. The taxonomies - with added functionality, such as inferencing - for these structures are called ontologies [1,2], and the success of the semantic web highly depends on the success of these ontologies. The reason ontologies are becoming popular

A. Laganà et al. (Eds.): ICCSA 2004, LNCS 3045, pp. 508–517, 2004.

is largely due to what they promise: a shared and common understanding of a domain that can be communicated between people and applications.

A major problem is that as an ontology grows bigger, user applications only require particular aspects of the ontology as they do not benefit from the plethora of semantic information that may be present in the ontology. However, using the ontology means that all the drawbacks from this extra information are encountered; complexity and redundancy rise, while efficiency falls. This brings with it a clear need to create a sub-ontology [3,4]. For instance, if a business (application) only concerns itself with the efficiency of the workers, there is no need to access the detailed product catalog. Extracting just the part that is needed offers a smaller, more efficient, simpler solution/ontology.

A lot of research in similar areas has been done (e.g. in [5,6,7,8]). Previous research by the authors pioneered in the specialized area of ontology extraction [9,10]. An extraction methodology, consisting of a number of optimization schemes, was introduced to meet the extraction requirements, and guarantee a high quality resulting sub-ontology. However, this extraction process often proves to be computationally expensive, because ontologies in realistic settings turn out to be very large. For instance, the Unified Medical Language Systems (UMLS) base ontology has more than 800,000 concepts (nodes) and more than 9,000,000 relationships between those concepts).

This work was done as a part of a bigger project [11] involving materialized sub-ontology extraction using distributed methods. Distribution not only makes the process faster, but more importantly also facilitates our envisaged application of the extraction process. Often, business organizations have a cluster-like setup of inter-connected workstations as opposed to a single *shared-memory*, High-Performance Computing (HPC) facility. One reason for this is that a *'Beowulf Class Cluster'* setup is easily affordable than a centralized HPC facility. It is this setup that we aim to leverage upon by implementing a distributed memory architecture for the sub-ontology extraction process. In this paper, we look at the issue of semantic completeness of an extracted sub-ontology implemented in a distributed environment. As with all stages, even this stage will be referred to as an optimization scheme, hence the name Semantic Completeness Optimization Scheme (SCOS).

2 Previous Work: MOVE

Figure 1 shows a schematic of the sequential extraction process called Materialized Ontology View Extraction (MOVE) [11]. The process begins with the import of the ontology externally represented using XML. The actual extraction process/execution of optimization schemes is initiated by way of *requirements specification* by a user or another application. In the sub-sections that follow, each of the main components illustrated in Fig. 1 will be discussed briefly.

The ***'ontology import layer'*** (component 1) is responsible for handling various ontology representation standards that the extraction process is supposed to be compliant with. This is currently achieved in MOVE by transforming

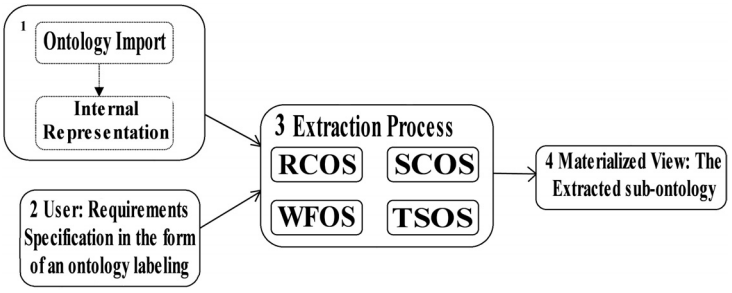

Fig. 1. The Sequential Extraction Process

the external representation of the ontology and its *meta-level* to a internal one that is specific to our implementation. It is necessary for user applications to use our import layer so as to be able to utilize the extraction algorithms. The *representation layer* maintains an object-oriented view of the ontology and its meta level. This facilitates easy extensibility as new ontology elements (new types) may easily be added in the ontology as well as its meta-level.

*'**Labeling**'* (component 2) of the base ontology facilitates user manipulation of the extraction process. The labeling may also be re-applied (i.e. modification of the user specified labeling) by the intermediate steps involved in the extraction process. This is the standard way different components of the extraction process (different extraction algorithms) may communicate with each other. Therefore, labeling is very crucial in the interaction between users & the extraction algorithms and algorithms amongst themselves. It allows a user to provide subjective information, pertaining to what must/must not be included in the target sub-ontology, on which the extraction process is based on. Moreover, an algorithm may work upon the labeling specified by the user, modify it in a certain way while preserving the semantics of the specification and pass it on to another algorithm within the extraction process. Currently, every ontological element may have a labeling of ***selected*** - must be present in the sub-ontology, ***deselected*** - must be excluded from the sub-ontology or *void* - the extraction algorithm is free to decide the respective elements inclusion/exclusion in the sub-ontology.

The *'**extraction process**'* (component 3) involves application of various optimization schemes that handle various issues pertaining to it such as ensuring consistency of initial requirements, well-formedness and deriving a sub-ontology that is highly qualitative in a sense that it is optimum and is the best solution to the users requirements. Note that the extraction process in not limited to optimization schemes currently being used in our framework. Also, it is possible that a particular scheme be completely left out of the it. Currently, the extraction process consists of Requirements Consistency Optimization Scheme (RCOS1-RCOS4), Semantic Completeness Optimization Scheme (SCOS1- SCOS3), Well Formedness Optimization Scheme (WFOS1-WFOS5) and Total Simplicity Optimization Scheme (TSOS1 - TSOS3). RCOS checks for the consistency of the user specified requirements for the target ontology and SCOS considers the complete-

ness of the concepts, i.e. if one concept is defined in terms of an another concept, the latter cannot be omitted from the sub-ontology without loss of semantic meaning of the former concept. It might be possible that the user requirements (labeling) is consistent. However, there might be statements that inevitably lead to a solution that is not a valid ontology. WFOS contains the proper rules to prevent this from happening. Applying TSOS to a existing solution will result in the smallest possible solution that is still a valid ontology.

The result of the extraction process is not just simply a extracted sub-ontology, but rather an extracted ***'materialized ontology view'*** (component 4) [9]. In the extraction process, no new information should be introduced (e.g. adding a new concept). However, it is possible that existing semantics are represented in a different way (i.e. a different view is established). Intuitively, the definition states that - starting from a base ontology - elements may be left out and/or combined, as long as the result is a valid ontology. In the process, no new elements should be introduced (unless the new element is a combination of a number of original elements, i.e. the *compression* of other elements). A *materialized* ontology view is required, as the resulting sub-ontology should be an independent ontology, i.e. *should* be a valid ontology even if the base ontology is taken away.

3 Semantic Completeness of a Sub-ontology

The idea of semantic completeness of an ontology can be interpreted in a number of ways. However, for the purposes to sub-ontology extraction, it amounts to the inclusion of the ***defining elements*** for the elements selected by the user by way of requirements specification. A defining element is a concept, relationship or attribute that is essential to the semantics of an another element of the ontology. A concept selected to be present in the sub-ontology would be semantically incomplete if its super-concept (the defining element in this case) is deselected at the same time. This could be further generalized into a situation where a set of elements are connected by a IS-A relationship unto any arbitrary depth. The scenario can only get more complex in the presence of more complex relationships such as multiple-inheritance, aggregation etc. The Semantic Completeness Optimization Scheme (SCOS) exists to guard against such inconsistencies. Below we present some notation consistent with [9], which is useful to define a common vocabulary pertaining to the ontological workload.

- $\delta_{\mathrm{B}}(\mathrm{b})$: Denotes a binary relationship between concepts
- $\delta_{\mathrm{C}}(\Pi_1(\mathrm{b}))$: First concept associated with $\delta_{\mathrm{B}}(\mathrm{b})$
- $\delta_{\mathrm{C}}(\Pi_2(\mathrm{b}))$: Second concept associated with $\delta_{\mathrm{B}}(\mathrm{b})$
- $\delta_{\mathrm{attr}}(\mathrm{t})$: Denotes a attribute-concept relationship
- $\delta_{\mathrm{C}}(\Pi_1(\mathrm{t}))$: A concept with associated attribute, ie: the concept in a $\delta_{\mathrm{attr}}(\mathrm{t})$
- $\delta_{\mathcal{A}}(\Pi_2(\mathrm{t}))$: An attribute with associated concept, ie:the attribute in a $\delta_{\mathrm{attr}}(\mathrm{t})$

Before we proceed with illustrating the distribution scheme, it is necessary that each of SCOS1-SCOS3 be defined, albeit informally for the purposes of

this paper. A formal introduction to SCOS (and the entire extraction process) along with a practical walk-through with intuitive examples can be found in [9]. SCOS1-SCOS3 are as follows: 1) **SCOS1**: If a concept is selected, all its super-concepts, and the inheritance relationships between the concepts and its super-concepts have to be selected. 2) **SCOS2**: If a concept is selected, all the aggregate part-of concepts of this concept, together with the aggregation relationship have to be selected as well. 3) **SCOS3**: If a concept is selected, all the attributes it possesses with a minimum cardinality other than zero and their attribute mappings should be selected as well.

4 SCOS: Proposed Distributed Implementation

SCOS1 and SCOS2 are conceptually similar with the difference that the former deals with a collection of inheritance relationships while the latter with a collection of aggregation relationships. This collection can be conceptualized as a forest of sparsely connected undirected graphs with the concepts representing the vertices and the relationships representing the edges of the graph. Such a conceptualization (& representation) using a graph-theoretic approach is optimal (& convenient) for purposes of distribution of SCOS. For example, consider checking the semantic for the completeness of a (potentially huge) set of concepts related by binary inheritance relationship, ie., SCOS1. If the set is to be partitioned & distributed to different processors so that SCOS1 may be run on each of the partitions in parallel, it is obviously desirable to allocate one connected component to each of the processors.

4.1 Problems with Graph Based Representation

A major problem with representing the data partitioning problem (for SCOS) in a graph-theoretic manner is that our underlying ontology representation is not graph theoretic. During ontology import, we construct a object oriented representation of the ontology as well as its meta-level. No structural information regarding the connectivity of the ontological elements is present. Since a ideal ontology would be massive in size and complex in structure, it would be optimal from a performance view-point that the graph based representation be constructed at the time of initial ontology import. Our object-oriented design represents a trade-off decision we took given the fact that other optimization schemes (such as RCOS) do not benefit from a graph based representation. So a graph based representation encompassing all different types of ontological elements would not be particularly useful.

4.2 Proposed Solution

Prior to data partitioning for SCOS, a graph based representation for elements specific to SCOS (binary inheritance & aggregation relationships) is constructed. We use the standard adjacency structure representation (comprised of adjacency

lists) to construct the ontoGraph. This involves additional work in the form of pre-processing of the SCOS workload to extract the concept set (ie. the vertices of the graph). This is necessary so as to construct the adjacency structure representation. Once the graph based representation is complete, we use a technique similar to a depth first search algorithm on the graph to get the set of connected components (called partitions hereafter) so as to schedule each of those for distribution to worker processors. Below, we discuss the three main steps, namely Ontology pre-processing, ontoGraph construction and Partition formation, involved in ontology pre-processing.

- **Ontology pre-processing:** The pre-processing phase basically involves constructing the vertex *set* so as to be used by the ontoGraph construction module. The input to this phase is the whole ontology. Processing begins by extracting the *list* of binary (inheritance & aggregation for SCOS1 & SCOS2 respectively) relationships from the ontology and inserting the elements related by each of the relationships in the list to a *set based container* thereby avoiding duplicates. Moreover, the unique vertices in the vertex set are keyed from 0 to N - 1, where N is the cardinality of the vertex set.
- **OntoGraph Construction:** As mentioned before, we represent the ontoGraph using the standard adjacency structure representation. The adjacency structure consists of a vector of lists of graph nodes. Each node in turn consists of other information such as an integral id of the ontology element is represents, a link to the element is represents etc. This ancillary information is necessary during the next phase, namely Partition Formation.
- **Partition Formation:** Partition formation in our case is equivalent to finding the different connected components in the ontoGraph. We currently use a technique similar to a depth first traversal (of the ontoGraph) to achieve this. The input to this phase is the ontoGraph whereas the result consists of a list of partitions. Here, one might get the impression that all the partitions need to be formed before any of them are assigned to worker processors. However in actuality, there is no reason to wait for the next partition to be generated before the current one is scheduled for distribution to a free processor as the partition sets are all going to be disjoint. As shall be illustrated later, we make use of asynchronous distribution primitives to assign the most recently generated partition to a free processor without waiting for the next one to be formed. This is advantageous as working out the semantic completeness (for the assigned partition) and formation of the next partition can proceed in parallel. The asynchronous nature of the primitives only adds to this optimality.

4.3 SCOS Distribution

For implementing the Requirements Consistency Optimization Scheme [9,11], we utilized a modified version of the classic task-farm model. SCOS essentially utilizes a similar distribution model with the exception that there is continued two-way interaction between the master & worker processors. Moreover, unlike

the RCOS distribution scheme, the worker processes do not need to post *updates* (requests for missing data) as they have all the data elements that would ever be needed to perform the most recently assigned task (ie: any of SCOS1-SCOS3). Also, data partitioning by the master & processing by the workers happens concurrently as the master dynamically creates partitions and assigns them to worker processes in a round-robin manner.

We use three asynchronous data distribution/result collection primitives namely ***gatherModifiedLabelingsFrom(...)***, ***recvPartition(...)*** and ***sendModifiedLabelings( )***. *recvPartition(...)* is used by the worker processes to receive the ontological workload that needs to be processed. Likewise, *gatherModifiedLabelingsFrom(...)* is used by the main processor to gather results from the worker processors, which they send using the *sendModifiedLabelings( )* primitive . As explained in section 2, this result takes the form of the modified labeling set. Note that it may be possible for the main processor to receive a 'semantic incompleteness' message and still get a modified labeling set. This is because following the rules for SCOS1-SCOS3, the worker processors attempt to make the extracted view as semantically complete as possible even if a 100% completeness is not possible.

Master processor execution consists of performing the necessary preprocessing of the ontology as explained in section 4.2. To re-iterate, it involves ontology initialization, extraction of the unique vertex & edge set, building the ontoGraph representation and perform the ontoGraph partitioning coupled with asynchronous distribution to the worker processors. Once the distribution is achieved, the only thing remains to be done for the master is collection and application of results to its solution set or the extracted view. As mentioned previously, irrespective of the results (ie: semantic completeness/incompleteness), the master always applies the labeling modifications worked out and serialized back

```
BEGIN Master
1. terminate = false;
2. resultsCounterSCOS1 = 0;
3. O = intialiseOntology();
4. V = preProcess(O);
5. G = buildOntoGraph(V);
6. P = partitionGraph(G);
7. partitionCount = |P| ;
8. while(terminate == false)
   {
      msg = getMessage();
      src = getSender();
      if(msg == SCOS1_SEMANTIC_COMPLETENESS)
        resultsCounterSCOS1++;
      else if(msg == SCOS1_SEMANTIC_INCOMPLETENESS)
        resultsCounterSCOS1++;

      modifiedLabelings = gatherModifiedLabelingsFrom(src);
      applyModifications(modifiedLabelings);

      if(resultsCounterSCOS1 == partitionCount)
        terminate = true;
   }
9. broadcastExitMessage(ALL_WORKERS);
END
```

```
BEGIN Worker
1. terminate = false;
2. while(terminate == false)
   {
      msg = getMessage();
      if(msg == SCOS_EXIT)
         teminate = true;
      else if(msg == SCOS1_PARTITION)
      {
         [δ_B(b)] = recvPartition(SCOS1_PARTITION)
         O = initLocalOntology([δ_B(b)]);
         Result = SCOS1_Single(O);
      }
      else if(msg == SCOS2_PARTITION)
      {
         [δ_attr(t)] = recvPartition(SCOS2_PARTITION)
         O = initLocalOntology([δ_attr(t)]);
         Result = SCOS2_Single(O);
      }

      sendMessage(Result);
      sendModifiedLabelings();
   }
END
```

Fig. 2. Master & Worker Processors

by the worker processes. Depending on the number of partitions scheduled to the worker processors, the master can figure out when results for each of them have been received and it is appropriate for the workers and itself to terminate.

Worker execution involves checking for a '*execution command*' from the main processor. Possible commands are to receive the workload pertaining to SCOS1-SCOS3 or actually execute SCOS1-SCOS3. Execution of any of the optimization schemes is followed by the sending back of results (modified labeling) and an indication of whether or not semantic completeness is possible for the most recently received partition. Note that the worker only terminates upon receiving an *'SCOS_EXIT'* message from the main processor.

4.4 Implementation

All implementation has been done using C++ on a Alphaserver SC supercomputer running Tru64 Unix 5.1. It has also been ported to a Linux Cluster environment with minimal modifications. Our distribution management component does not directly tackle issues pertaining to the cluster architecture, processor initialization etc. Instead, the *Messaging Passing Standard* [12], which encapsulates such architecture specific details and provides high level message-passing primitives suitable for distributed systems, has been utilized. As such, porting to other environments should not involve anything more than a recompilation on the target platform. Also, note that although homogeneous computing elements are being utilized currently, this is not a requirement for the implementation. Any distributed architecture is good enough as long as it supports the MPI message passing standard. It is up to the standard to handle the underlying architectural details pertaining to the cluster setup.

5 Evaluation

This section analyzes the results obtained from the distribution of the SCOS processing. Five different ontologies were used for testing the performance of SCOS. Initialization of the ontology is a constant time operation. As such the time taken to load each ontology, its meta-ontology and the associated user requirements (labeling) has been excluded. The results shown only include the time that the SCOS processing and distribution were active. The results have been split into two graphs to make it easier to show the similarities and differences between the five different sized ontologies. Fig. 3 - graph(a) shows that small ontologies (1-2000 concepts) with a greater number of processors, doesn't improve the overall time. From 3000 concepts and above, using a greater number of processors does speed up the time taken to complete the work. However, the larger ontologies in graph(b), seem to flatten off as the number of processors increase. This is where the added cost of communication for the extra processors out-weighs the amount of work left to be distributed. From these graphs then, we can suggest that for a certain ontology size, SCOS should employ a specific number of processors to do the work to obtain the most efficient use of resources.

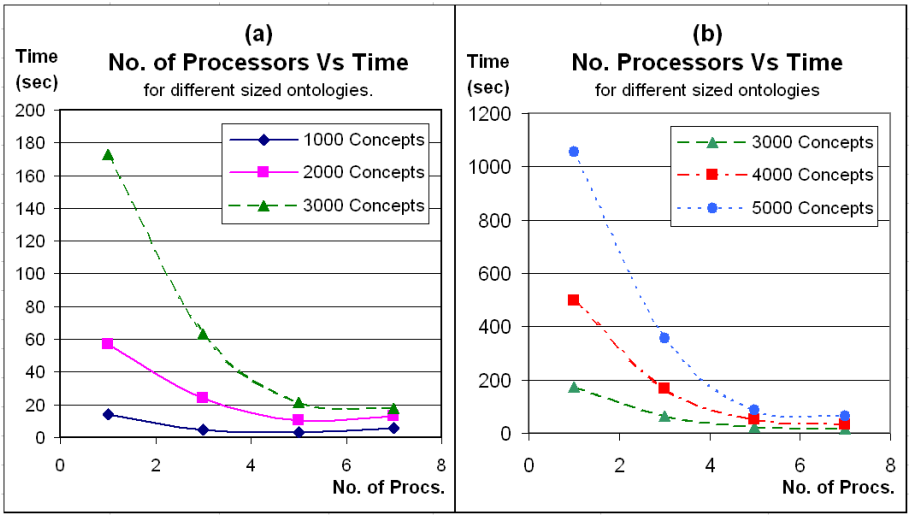

Fig. 3. SCOS Performance Results

6 Conclusion and Future Work

SCOS currently consists of three sub-schemes SCOS1-SCOS3, which handle various issues pertaining to semantic completeness. As mentioned previously, we have defined semantic completeness to be the inclusion/exclusion of certain *defining elements* for the ontological elements selected/deselected by the user or other application. Obviously, this notion of semantic completeness could be expanded and new rules could be specified by other researchers in the ontology domain. However, the distributed architecture that we have proposed and implemented is general enough to be used with other ontology based applications. The plugin based design of the optimization schemes as well as the distribution primitives facilitates seamless integration with other ontology applications.

The use of dynamic process management capability needs to be implemented into the system. Until the development of RCOS [11], the need for such capability within the framework did not arise as RCOS merely consists of partitioning the data to be processed based on the number of processors available. However, in the case of SCOS, the number of partitions generated is a property of the structure/connectivity of the OntoGraph. As such, a capability to allocate an optimal number of processors based on the number of data partitions will be necessary for optimized execution. Currently, we use a very coarse-grained distribution scheme. It is possible to utilize more sophisticated distribution schemes given the fact that there are no constraints as to the order of execution of functionally independent optimization schemes. Also, more fine-grained solution could consist of parallelization of individual sub-schemes within a particular optimization scheme. However, these enhancements would only be justified if optimal performance of the extraction process is a absolute

necessity. As previously mentioned, our research and the resulting distributed architecture is strongly driven by our envisaged application of the extraction process in a distributed business environment.

Acknowledgment. This work has been financially supported by Victorian Partnership For Advanced Computing (VPAC) Expertise Grant Round 5, No:EPPNLA090.2003. All implementation was done using VPAC's supercomputing facilities.

References

1. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. In: Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers, Deventer (1993)
2. Guarino, N., Carrara, M., Giaretta, P.: An ontology of meta-level categories. In: KR'94: Principles of KRR. Morgan Kaufmann (1994)
3. Wouters, C., Dillon, T., et. al: A practical walkthrough of the ontology derivation rules. In: Proceedings of DEXA 2002, Aix-en-Provence (2002)
4. Spyns, P., Meersman, R., Mustafa, J.: Data modelling versus ontology engineering. SIGMOD (2002)
5. Guarino, N., Welty, C.: Evaluating ontological decisions with ontoclean. Communications of the ACM **45** (2002)
6. Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D.: Ontology versioning and change detection on the web. In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Volume 2473 of LNAI., Spain, Springer Verlag (2002) 197–212
7. McGuinness, D.L., et. al.: An environment for merging and testing large ontologies. In: Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning, San Francisco, Morgan Kaufmann (2000)
8. Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. Technical Report SMI-2002-0926, Stanford Medical Informatics (2002)
9. Wouters, C., Dillon, T., Rahayu, W., et. al: A practical walkthrough of the ontology derivation rules. DEXA2002 (2002) 259–268
10. Wouters, C., Dillon, T., Rahayu, W., et. al: A practical approach to the derivation of materialized ontology view. In: Web Information Systems. Idea Group Publishing (2004)
11. Bhatt, M., Flahive, A., Wouters, C., Rahayu, W., Taniar, D., Dillon, T.: A Distributed Approach to Sub-Ontology Extraction. In: Proceedings of the Eighteenth International Conference on Advanced Information Networking and Applications (AINA'04), Fukuoka, Japan (2004)
12. William Gropp, Ewing Lusk, A.S.: Using MPI: Portable Parallel Programming with the Message-Passing Interface. Second edn. MIT Press (1999)