A Distributed Approach to Sub-Ontology Extraction

Mehul Bhatt,
Andrew Flahive, Carlo Wouters, Wenny Rahayu
La Trobe University, Australia
{mbhatt, apflahiv, cewouter, wenny}
@cs.latrobe.edu.au

David Taniar
Monash University, Australia
David.Taniar@infotech.monash.edu.au
Tharam Dillon
University of Technology Sydney, Australia
tharam@it.uts.edu.au

Abstract

The new era of semantic web has enabled users to extract semantically relevant data from the web. The backbone of the semantic web is a shared uniform structure which defines how web information is split up regardless of the implementation language or the syntax used to represent the data. This structure is known as an ontology.

As information on the web increases significantly in size, Web ontologies also tend to grow bigger, to such an extent that they become too large to be used in their entirety by any single application. This has stimulated our work in the area of sub-ontology extraction where each user may extract optimized sub-ontologies from an existing base ontology.

Sub-ontologies are valid independent ontologies, known as materialized ontologies, that are specifically extracted to meet certain needs. Because of the size of the original ontology, the process of repeatedly iterating the millions of nodes and relationships to form an optimized sub-ontology can be very extensive. Therefore we have identified the need for a distributed approach to the extraction process. As ontologies are currently widely used, our proposed approach for distributed ontology extraction will play an important role in improving the efficiency of information retrieval.

1. Introduction

The next generation of the internet is the *semantic web*. It 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[3, 4], and the success of the semantic web highly depends on the success of these ontologies. Ontology user groups and communities need to agree on proper ontology standards before they can be put to good use. This was made clear from the initial definition of an ontology as a *shared* conceptualization of some domain [3]. Getting large groups to agree on an ontology standard means that a compromise is needed to ensure world wide use of ontologies. Ontologies grow very large as a result of attempts to model certain domains accurately and completely, leading to a number of problems.

A major problem is that as an ontology grows bigger, user applications begin to only require particular aspects of the whole ontology. They do not benefit from other semantic information. Using the whole ontology means that drawbacks from this redundant information are encountered; complexity and redundancy issues rise, while efficiency issues fall. This demonstrates a clear need to create sub-ontologies[14, 10]. For instance, if a business only concerns itself with worker information, there is no need to access the detailed product catalog information. Extracting just the worker information, offers a smaller, simpler and more efficient ontology solution.

There is research in similar areas[5, 6, 7, 8], and also there is research pioneered by the authors in the specialized area of ontology extraction[14, 15, 16]. Optimization schemes were introduced to meet requirements to guarantee a high quality sub-ontology. However, this extraction process, proves to be computationally expensive, in part because of large ontologies (The Unified Medical Language Systems (UMLS) [9] ontology has more than 800,000 concepts and more than 9,000,000 relationships), but also because of the many optimization schemes have to be applied.

This paper introduces a distributed approach to the sub-ontology extraction process. Distribution not only makes the process faster, but also conforms to our envisaged application. The model presented in this paper will benefit a lot of areas involved in ontology manipulation and extraction as well as general workload distribution.

The rest of this paper is as follows: Section 2 introduces the sequential extraction process we call *MOVE* - M*aterialized* O*ntology* V*iew* E*xtractor*. Section 3 presents

Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04)
0-7695-2051-0/04 $ 20.00 © 2004 IEEE

the distributed version of MOVE, by stepping through one of the basic optimization schemes involved in the extraction process - The Requirements Consistency Optimization Scheme (RCOS). Section 4 evaluates the distributed implementation, and Section 5 concludes the paper.

2. MOVE: Materialized Ontology View Extraction

The architectural framework for the whole process of extracting materialized ontologies, is shown in Figure 1. Throughout this paper we refer to the result of an extraction process as a sub-ontology or a materialized ontology view. Intuitively, and for the course of this paper, these can be defined as a new version of the ontology, where no new semantic information has been added but semantic information has been thoughtfully left out. Detailed definitions are given in [15, 16], but are irrelevant here.

The process begins with the import of an ontology (i.e. constructing an internal memory representation of the ontology), which is represented using an ontology standard such as OWL [11], DAML-OIL [1] or Ontolingua [2]. Also imported into the system is the user (or application) requirements and specifications. This is followed by the execution of the optimization algorithms that finally produce the extracted sub-ontology. Next we briefly discuss the major components of the system which are illustrated in Figure 1.

2.1. Ontology Import Layer

The *import layer* (component 1) is responsible for handling the various ontology representation standards. This is achieved in MOVE by transforming the external representation of the ontology and its *meta-level* to an internal one that is specific to our implementation. The Meta-level consists of type-information pertaining to the various elements in the ontology. If PERSON is an element in an ontology, the meta-level then holds the type of the element, say a CONCEPT. So a PERSON has a meta-type of CONCEPT.

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 be added to the ontology as well as its meta-level.

2.2. Labelling - Requirements Specification

'Labelling' (component 2) of the base ontology facilitates the manipulation during the extraction process. The labelling may be adjusted, from what the user specified, due to the requirements of the extraction process. This is the standard way that the different components of the extraction process (different extraction algorithms) communicate with each other. Therefore, labelling is crucial to the interaction between users and 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. Moreover, an algorithm may work on the labelling specified by the user, modify it and still preserving the semantics of the specification before passing it to the next algorithm in the extraction process.

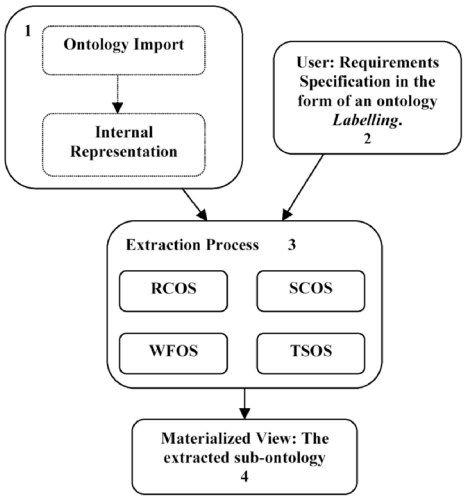

Figure 1. The Sequential Extraction Process

Every ontological element may have a labelling 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.

2.3. The Extraction Process

The *'Extraction Process'* (component 3) involves various optimization schemes that handle several issues pertaining to it, such as to ensuring the consistency of initial requirements. This maintains the semantic completeness, well-formedness and derives a sub-ontology that is highly qualitative, in a sense that it is optimal, and is the best solution based on the users requirements. The extraction process is not limited to the optimization schemes that are being used in the framework. Schemes can be added or removed based on the users requirements of the resulting sub-ontology. Below, we present a short discussion on each of the optimization schemes currently used in MOVE.

- **Requirements Consistency Optimization Scheme** (RCOS): RCOS checks for the consistency of the

user specified requirements for the target ontology. Currently, RCOS itself is a combination of four sub-schemes that check for various forms of consistency. We will cover this optimization scheme in greater detail in Section 3.

- **Semantic Completeness Optimization Scheme** (SCOS): SCOS considers the completeness 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.
- **Well Formedness Optimization Scheme** (WFOS): This optimization scheme contains the proper rules to check that the new sub-ontology is a valid ontology.
- **Total Simplicity Optimization Scheme** (TSOS): Applying TSOS to a solution will result in the smallest possible solution that is still a valid ontology.

2.4. Materialized View

The result of the extraction process is not simply just an extracted sub-ontology, rather an extracted *'materialized ontology view'* (component 4)[14]. In the extraction process, no new information is 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 elements may be left out and/or combined, as long as the result is a valid ontology. No new elements are introduced, unless the new element is a combination of a number of original elements (i.e. the *compression* of two or more elements). A materialized ontology view is required, as the resulting sub-ontology is a complete independent ontology.

3. Proposed Distributed Extraction Method

The complexity and size of the base ontology make the extraction process computationally extensive. However, the sequential extraction process, as illustrated in Section 2, lends itself to easy distribution. For instance, most of the intermediate extraction algorithms do not distinguish between the complete centralized ontology and a scaled down or partitioned version of it. Therefore the same extraction algorithm may be applied to both sequential as well as distributed versions, albeit with some modifications.

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 cheaper than a centralized HPC facility. It is this reason that we aim to implement on a distributed memory architecture for the extraction process described in Section 2.

3.1. Distribution Scheme

The core of the entire extraction process is the optimization scheme execution. As mentioned in the previous section, the optimization schemes are the modules within our framework which deal with matching user requirements and specifications to a base ontology. And then to extract the required concepts and relationships based on these specifications. Depending on the user's preference, optimization schemes can be tailored to meet certain criteria, such as requirement consistency (RCOS) or semantic completeness (SCOS) of the derived ontology, etc. Different optimization schemes can be used together or independently.

This paper focuses on the *Requirements Consistency Optimization Scheme* (RCOS). RCOS ensures that the requirements as expressed by the user (or any other optimization scheme) are consistent and correct, i.e. there are no contradictory statements in the *labelling,* as set by the user. The RCOS rules deal with, for example, where users have missed concepts or have selected attributes but not the concept that owns the attribute. RCOS It is made up of four rules, which without any implicit ordering or execution priority, we denote as RCOS1-RCOS4. The rules are provided below, a more formal introduction to RCOS (and the entire extraction process) can be found in [14].

- **RCOS1**: This rule stipulates that *if a binary relationship between concepts is* selected *by the user to be present in the target ontology*, *the two concepts that the relationship associates cannot be* disqualified/deselected *from the target ontology*.
- **RCOS2**: Similar to RCOS1 with the difference being that it is applied on an attribute mapping relationship instead. This rule enforces the condition that *if an attribute mapping has a* selected *labelling, the associated attribute and the concept that it is mapped onto must be* 'selected' *to be present in the target ontology*.
- **RCOS3**: Stipulates that *if an attribute mapping has a* deselected *labelling, its associated attribute must be disqualified from the target ontology*. No contradicting preferences are allowed between attribute mappings and their associated attributes. RCOS3 together with RCOS2 imposes this condition.
- **RCOS4**: RCOS4 is relatively more complex than each of RCOS1-RCOS3. We utilize the notion of a ***Path***, for illustrating RCOS4. Paths are very important in the specification of ontology views. They provide seemingly new relationships that are semantically correct. A path between two concepts is defined as a chain of relationships and concepts that connects those concepts.

  From an RCOS4 viewpoint, the emphasis of a path lies on the first and last concepts, as these are the two

that are connected by the path. RCOS4 imposes a condition that *If an attribute is selected, but the concept it 'belongs' (mapped) to is deselected, there must exist a path from the attribute to another concept that is not deselected. Moreover, the path can only contain relationships with a label other than 'deselected'.*

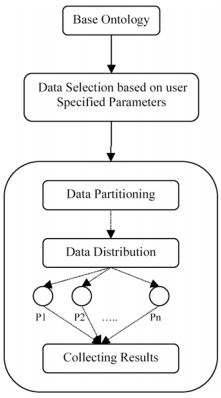

Figure 2. Task-Farm Distribution Model

Figure 2 illustrates the classic task-farm model that is being utilized for the distribution of RCOS. From Figure 3 it implies that the implementation of RCOS rule 1-3 is straight forward. However, the complexity actually lies in the sheer size of the workload that needs to be processed for ensuring consistency. One can also draw the obvious parallels that would exist in a distributed implementation of RCOS1-RCOS3. Hence, we refrain from demonstrating the distribution schemes for each of RCOS1-RCOS3 separately. Instead, we illustrate the general distribution scheme used by us (based on Fig. 2) for RCOS1-RCOS3.

Some Notation: Below we present some notation consistent with [14]. These notations are used in the pseudo code associated with the different optimization schemes.

1. δ_{B}(b): a binary relationship between concepts
2. $\delta_{\mathrm{C}}(\Pi_1(\mathrm{b}))$: First concept associated with δ_{B}(b)
3. $\delta_{\mathrm{C}}(\Pi_2(\mathrm{b}))$:Second concept associated with δ_{B}(b)
4. δ_{attr}(t): Denotes a attribute-concept relationship
5. $\delta_{\mathrm{C}}(\Pi_1(\mathrm{t}))$: A concept with associated attribute
6. $\delta_{\mathcal{A}}(\Pi_2(\mathrm{t}))$: An attribute with a associated concept
7. $\delta_{\mathrm{B}}(\mathrm{b_i})$, i ϵ [0, N]: Many binary relationships linked to make a path.

Let **O** denote the Ontology and ***filter(type, label)*** be an operation defined over it such that it returns a collection of elements, $\vec{e}_{type}$ of the type given by '***type***' such that the label of each of those elements matches '***label***'. Similarly, let ***partition***$(\vec{e}_{type})$ be a generic operation that partitions the $\vec{e}_{type}$ according to some data *partitioning scheme*. If N computing elements are available (numbered 1 to N), the result of ***partition***$(\vec{e}_{type})$ will be N distinct $\vec{e}^{\,i}_{type}$ where i represents the number of the computing element to which the respective partition has been assigned. Finally, let ***distribute and gather*** be *asynchronous* data distribution and result collection primitives.

```
1. Initialize O;
2. e⃗_type = filter(type, label);
3. [e⃗^i_type]  = partition(e⃗_type);
4. for(i = 1 To N)
       distribute(e⃗^i_type, i);
5. set an execution barrier
6. for(i = 1 To N)
       gather(e⃗^i_type, i);
7. merge results.
```

Figure 3. RCOS1-RCOS3 Distribution

Depending on which rule is being implemented, the parameters to the filter operation change accordingly. However, the rest of the algorithm is exactly the same for each of RCOS1-RCOS3. Currently, we adopt a equal workload distribution scheme. So the partitioning scheme divides the entire ontological workload into N distinct partitions, where N computing elements (processors) are available for use.

```
1. Initialize O;
     terminate = false; k = 0;
2. e⃗_type = filter(type, label);
3. [e⃗^i_type]  = partition(e⃗_type);
4. for(i = 1 To N)
    distribute(e⃗^i_type, i);
5. while(terminate != true)
  {
    src = readMessage;
    if(readMessage ==
                REQUEST_UPDATE)
    {
      e⃗^i_type = update(O);
      distribute(e⃗^i_type, src);
    }
    else if(readMessage ==
                COLLECT_RESULT)
    {
      gather(e⃗^i_type, src);
      ++k;
    }
    if(k == workerCount)
      terminate = true;
  }
```

Figure 4. RCOS4 Distribution

As mentioned before, RCOS4 utilizes an extended model to the one depicted in Figure 4. For RCOS4, the main (distributing process) and the worker processes follow a bi-

directional communication protocol consisting of *update requests* (sent by the worker processes) and *ontology updates* sent back by the main process. The *findSolution* algorithm (see RCOS4 pseudo code) works by traversing the ontology, trying to find a concept that is not deselected. This traversal may span parts of the ontology which are not *locally* present with the worker processor. The worker processors keeps track of such a situation and asks for an *update* from the main processor. The updateRequest-updateReceive protocol continues as long as the worker processes do not reach a dead end or a best path is found, which is then returned as the locally found solution. Notice that RCOS4 does not involve any merging of results like RCOS1-RCOS3. Every solution returned by any of the worker processes will be complete in itself.

3.2. Implementation

3.2.1. Plugin Architecture for Extraction Algorithms Our implementation consists of two main components. One pertaining to ontology representation and materialized view extraction and the other pertaining to management of the distributed extraction process. We adopt a plugin-based architecture so as to facilitate seamless integration of both the components into other ontological applications. It is possible for a user application to incorporate our functionality in the form of a plugin. Moreover, each phase within the extraction process, consisting of various optimization schemes, has been implemented independent of other phases. Therefore, applications importing our functionality are not compelled to use the entire extraction process. Depending upon intended use, any of the optimization schemes may be used in isolation.

3.2.2. Workload Distribution The ontological workload distribution pertaining to all of the optimization schemes turns out to be fairly standard. For instance, almost all optimization schemes involve the same data partitioning algorithms, distribution primitives etc. Only the rules that qualify data selection (For example, for partitioning purposes) involve minor modifications. There are two perceived advantages offered by this approach:

1. Firstly, it facilitates our own development involving distribution of ontology related workload.
2. Secondly, our distributed approach may be replicated by users in other innovative application domains as the distribution component encapsulates the classic task-farm distribution model, data partitioning and distribution primitives, as well as facilities the merging back of results.

3.2.3. Implementation Environment All implementation has been done using C++ on an Alpha server SC supercomputer running Tru64 Unix 5.1. It has also been ported to a Linux Cluster environment. Our distribution management component does not directly tackle issues pertaining to the cluster architecture, processor initialization etc. Instead, the Messaging Passing Standard[12, 13], that 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.

4. Evaluation

To effectively evaluate the distributed section of the program, a model was constructed to demonstrate the efficiency of the program relative to different sized ontologies. The graph has been properly scaled so that the different sized ontologies could be evaluated easily on the one graph (Figure 5). The graph shows the effectiveness of using multiple processors for different sized ontologies. It shows how efficiently each size of ontology is handled in the distribution. As can be seen, the distribution is not very efficient for ontologies that are less than 10,000 concepts. This is because the cost of employing a number of processors is to large for the relatively small amount of work required to be done. However, for ontologies that contain larger than 10,000 concepts, the optimum number of processors to employ increases.

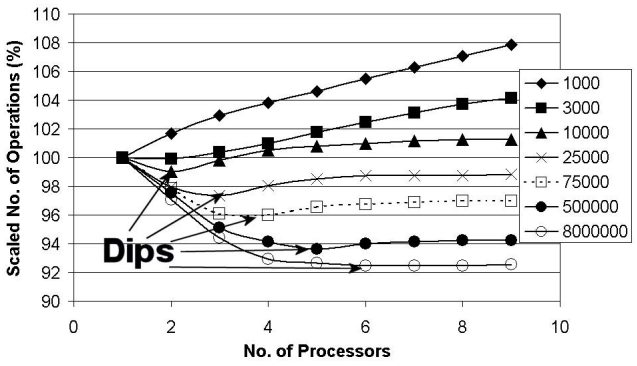

Figure 5. Scaled Performance Comparison

The dip in the plots (plots with greater than 10,000 concepts) show the points where the optimum number of processors to employ is found. For example, if we were using an ontology with 75,000 concepts, it would be best to employ four processors, if we used any less or any more then the total number of operations would increase, leading to an inefficient use of the processors.

Increasing the number of processors past the dip point, for any sized ontology, the number of operations taken to complete work increases due to the extra operations re-

quired to employ the extra processors and distribute the work. These sections in the plots, show that processors are wasting precious operations and not doing any extra work. Basically the small amount of work required to be done doesn't warrant the employment of extra processors.

Based on these results, it is proposed that the distribution follow this basic table (Table 1) to ensure that the work being completed uses the most optimal number of processors to minimize the overall number of operations performed.

Size of Ontology	Optimum No of Processors
less than 10,000	1 Processor
10,000 - 25,000	2 Processors
25,000 - 75,000	3 Processors
75,000 - 500,000	4 Processors
500,000 - 8,000,000	5 Processors
8,000,000 plus	6 Processors

Table 1. Optimal Processors

For ontologies that have less than 10,000 concepts, the system should only employ one processor to complete the work to be at it most efficient. If more processors were to be used then they would be wasted and the program would actually take longer to run. For ontologies between 10,000 concepts and 25,000 concepts, the system should use two processors. The same goes for ontologies of sizes between 25,000 and 75,000, 75,000 and 500,000, and 500,000 and 8,000,000 concepts. For ontologies that had more than 8,000,000 concepts there seemed to be no real benefit for using more than six processors in this example.

5. Conclusion and Future Work

In this paper we have demonstrated a novel approach for sub-ontology extraction from a large scale base ontology. We have developed and implemented a distributed approach for the execution of the extraction process. The requirement consistency sub-ontology optimization scheme (RCOS) was divided into four modules (RCOS1 to RCOS4) to distinguish the optmization scheme rules.

The evaluation shows how the performance of the extraction process can be improved in a multi-processor distributed environment by employing an appropriate number of worker processors. The result of this work will be beneficial for other web applications that utilize large ontologies such as bioinformatics area, data warehousing, and medical sciences. By using our extraction process and distributed algorithms, an application can extract the necessary concepts and relationships from the base ontology and work independently on them.

6. Acknowledgment

The research presented in this paper has partly 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] T. Berners-Lee and Al. Reference description of the daml+oil ontology markup language, 2001.

[2] T. R. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical report, Knowledge Systems Laboratory, Stanford University, 1992.

[3] T. R. Gruber. 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.

[4] N. Guarino, M. Carrara, and P. Giaretta. An ontology of meta-level categories. In *KR'94: Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann, 1994.

[5] N. Guarino and C. Welty. Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 45, 2002.

[6] M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. 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*, pages 197–212, Spain, 2002. Springer Verlag.

[7] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, pages 483–493, San Francisco, USA, 2000. Morgan Kaufmann.

[8] N. F. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. Technical Report SMI-2002-0926, Stanford Medical Informatics, 2002.

[9] U. S. N. L. of Medicine. Unified medical language system, 2003. http://www.nlm.nih.gov/research/umls/.

[10] P. Spyns, R. Meersman, and J. Mustafa. Data modelling versus ontology engineering. *SIGMOD*, 2002.

[11] W3C. Owl web ontology language 1.0 reference, July 2002.

[12] A. S. William Gropp, Ewing Lusk. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, second edition, 1999.

[13] R. T. William Gropp, Ewing Lusk. *Using MPI-2: Advanced Features of theMessage-Passing Interface*. MIT Press, 1999.

[14] C. Wouters, T. Dillon, W. Rahayu, and E. Chang. A practical walkthrough of the ontology derivation rules. In *Proceedings of DEXA 2002*, Aix-en-Provence, 2002.

[15] C. Wouters, T. Dillon, W. Rahayu, E. Chang, and R. Meersman. A practical approach to the derivation of materialized ontology view. In *Web Information Systems*. Idea Group Publishing, to appear January 2004.

[16] C. Wouters, T. Dillon, W. Rahayu, R. Meersman, and E. Chang. Transformational processes for materialized ontology view specification. *(submitted for publication)*, 2003.