Generating Ontologies via Language Components and Ontology Reuse

Yihong Ding[1], Deryle Lonsdale[2], David W. Embley[1], Martin Hepp[3], and Li Xu[4]

[1] Department of Computer Science, Brigham Young University, U.S.A.
{ding,embley}@cs.byu.edu
[2] Department of Linguistics, Brigham Young University, U.S.A.
lonz@byu.edu
[3] Digital Enterprise Research Institute (DERI), University of Innsbruck, Austria
martin.hepp@deri.org
[4] Department of Computer Science, University of Arizona South, U.S.A.
lxu@email.arizona.edu

Abstract. Realizing the Semantic Web involves creating ontologies, a tedious and costly challenge. Reuse can reduce the cost of ontology engineering. Ontologies already created in recent Semantic Web research provide useful input for ontology reuse. However, automated ontology reuse remains underexplored. This paper presents a generic architecture for automated ontology reuse. With our implementation of this architecture, we show the practicality of automating ontology generation through ontology reuse. We experimented with a large generic ontology (MikroKosmos) as a basis for automatically generating domain ontologies that fit the scope of sample natural-language web pages. The results were encouraging, resulting in five lessons pertinent to future automated ontology reuse study.

1 Introduction

Ontology construction is a central research issue for the Semantic Web. Ontologies provide a way of formalizing human knowledge to enable machine interpretability. Creating ontologies from scratch is, however, usually tedious and costly. When the Semantic Web requires ontologies that express Web page content, the ontology engineering task becomes too expensive to be done manually. Many Semantic Web ontologies may have overlapping domain descriptions because many Web sites (or pages) contain information in common domains. It is inefficient to redo ontology engineering for pre-explored domains. These issues illustrate the importance of automated ontology reuse for the Semantic Web.

Ontology reuse involves building a new ontology through maximizing the adoption of pre-used ontologies or ontology components. Reuse has several advantages. First, it reduces human labor involved in formalizing ontologies from scratch. It also increases the quality of new ontologies because the reused components have already been tested. Moreover, when two ontologies share components through ontology reuse, mapping between them becomes simpler because

mappings between their shared components are trivial. One can also simultaneously update multiple ontologies by updating their commonly reused components. Hence ontology reuse also improves the efficiency of ontology maintenance.

Despite the many advantages of (automated) ontology reuse, the topic is not well explored in the literature. There are many reasons for this. Before the advent of the Semantic Web, few ontologies existed. Due to the difficulty of constructing ontologies, as well as to the challenges of using ontologies in applications, researchers were less interested in ontology development. With the advance of Semantic Web technologies, the number of ontologies has significantly increased recently. When the use of ontologies in Semantic Web applications improves system performance, more people will realize the benefit of using ontologies. In the meantime, most existing ontologies are hard to reuse. The benefits of manual ontology reuse are often unclear since the overhead of seeking and understanding existing ontologies by humans may be even greater than simply building an ontology from scratch. At the same time, many existing ontologies simply do not support effectively automated ontology reuse. The corresponding information in these ontologies is hard to retrieve for automated ontology reuse.

The work we describe below[5] offers three contributions for automated ontology reuse. First, though, we sketch the state of the art in ontology reuse (Section 2). We then present our generic ontology reuse architecture and our implementation (Section 3). Next, we discuss our experimental results obtained by using our implementation on real-world examples, as well as five lessons we have learned from this work (Section 4). Finally, we mention possible future directions (Section 5).

2 Related Work

Ontology reuse has been studied for years. Most of the earlier research focuses on the study of reusable ontology repositories. In 2001, Ding and Fensel [7] surveyed these earlier ontology libraries. Due to the lack of ontologies, however, very few studies on practically reusing ontologies exist prior to this survey. Uschold and his colleagues [13] presented a "start-to-finish process" of reusing an existing ontology in a small-scale application. According to the authors, the purpose was a "feasibility demonstration only." They concluded that reusing an ontology was "far from an automated process" at that time.

With the growth of semantic web research, more and more ontologies have been created and used in real-world applications. Researchers have started to address more of the ontology reuse problem. Typically, there are two strands of study: theoretical studies of ontology reusability [2, 4, 8], and practical studies of ontology reuse [1, 11, 12]. Previous studies of ontology libraries showed that it was difficult to manage heterogeneous ontologies in simple repositories. Standardized modules may significantly improve the reusability of ontologies. One

[5] This work was partially funded under National Science Foundataion Information and Intelligent Systems grant IIS-0414644. See also www.deg.byu.edu.

major purpose of modular ontology research concerns the reusability of ontologies [2, 4, 8]. There are, however, fewer ontology reuse studies quantifying how modular ontologies may improve the efficiency of ontology reuse. Hence one of our purposes is to argue for the use of modular ontologies in real-world, automated ontology reuse experiments.

Meanwhile, there are also several studies on practical ontology reuse. Noy and Musen [11] introduced "traversal views" that define an ontology view, through which a user can specify a subset of an existing ontology. This mechanism enables users to extract self-contained portions of an ontology describing specific concepts. Stuckenschmidt and Klein [12] described another process for partitioning very large ontologies into sets of meaningful and self-contained modules through a structure-based algorithm. Alani and his colleagues [1] coined a new term for reusing existing ontologies: ontology winnowing. The intuition of their research is that individual semantic web applications more profitably use smaller customized ontologies rather than larger general-purpose ontologies. They therefore described a method for culling out—which they called winnowing—useful application-specific information from a larger ontology.

A common implicit assumption in all these practical ontology reuse studies is that source ontologies must be reusable for a target domain. Although this assumption simplifies the problem, it does not address the general situation. Besides our work, to the best of our knowledge, the only research that has addressed (albeit implicitly) the domain-specific ontology reuse problem is by Bontas and her colleagues [3]. Their case studies on ontology reuse identified difficulties due to end-user unfamiliarity with the complex source structure. Although this assessment is reasonable, we found a further reason for the difficulty they encountered. Even though source ontologies often declare a target domain, the corresponding information is irretrievable for automated ontology reuse. This is the real bottleneck for automated ontology reuse.

3 Automated Ontology Reuse

Figure 1 shows our generic architecture for automated ontology reuse. An automated ontology reuse procedure should take at least two inputs: natural language (NL) documents and source ontologies. NL documents express the projecting domain and they can encompass different types. Typical NL documents could include collections of competency questions [14] or collections of sample Web pages [5].

In this architecture, ontology reuse consists of three sequential steps: concept selection, relation retrieval, and constraint discovery. These correspond to the three fundamental components in ontologies: concepts, relationships, and constraints. The concept selection process identifies reusable ontology concepts from source ontologies based on the descriptions in NL documents. NL documents must contain sufficient information for a system to identify all the necessary domain concepts. The identification methodologies vary with respect to different types of NL documents. The relation retrieval process retrieves relation-

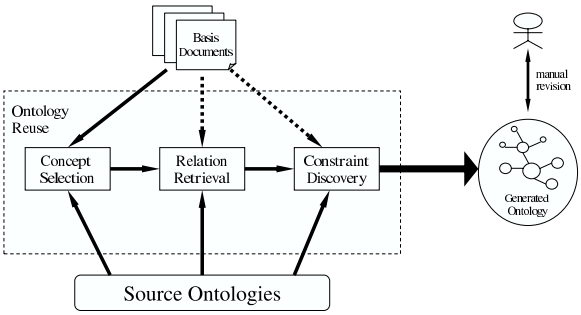

Fig. 1. Generic Architecture of Automated Ontology Reuse.

ships among selected concepts from the previous step. These relationships can be automatically gathered from source ontologies or perhaps even (optionally) recoverable from the NL documents. Figure 1 represents these optional requirements with dotted lines. The *constraint discovery process* discovers constraints for previous selected concepts and relationships. An ontology reuse system should be able to gather existing information about constraints from source ontologies or even perhaps from NL documents.

After these three sequential steps, the system composes the selected concepts, relationships, and constraints together into a unified ontology. Human experts then inspect and revise these auto-generated ontologies.

We have implemented a prototype automated ontology reuse system based on this generic architecture. Our system reuses existing ontologies to create small domain ontologies within the scope of describing individual Web pages. In the rest of this section we describe the system in fuller detail.

Preparation of Input We first take a small set of sample Web pages as input NL documents, and pre-process them to focus on the domain of interest (i.e. removing advertisement and side bars). Only the main body of each page remains, which constitutes the focus of interest for readers.

Proper preparation of source ontologies is essential for ontology reuse automation. Poorly integrated source ontologies create a very complex ontology integration problem during final composition of the output ontology. Two options exist: either we can directly adopt a single large-scale ontology, or we can manually pre-integrate several small ones. For simplicity, we chose the first option (and will discuss the second option later). Specifically, we adopted the MikroKosmos (μK) ontology [10], a large-scale ontology containing more than 5000 hierarchically-arranged concepts (excluding instances). These concepts cover various domains, a desideratum for flexible experimentation. The μK ontology has an average of 14 inter-concept links per node, providing rich interpretations for the defined concepts.

To automate our ontology reuse process, we pre-integrated the leaf concepts from the μK ontology with external lexicon dictionaries and declarative data

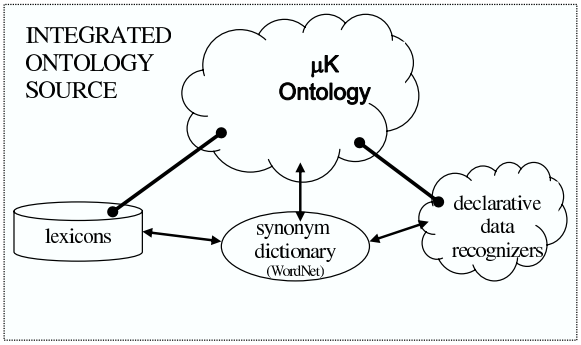

Fig. 2. Pre-Integrated Source Ontology.

recognizers, as Figure 2 shows. These augmentations are essential for automated ontology-concept recognition. Most of these lexicons and data recognizers are collected from the Web. For example, for the ontology concept CAPITAL-CITY we used a web browser to locate lists of all the capital cities of the independent countries in the world. Since we collected information from varied resources, we found that synonym identification became critical for the performance of ontology reuse. We therefore adopted WordNet[6] for our synonym resource.

Although this source ontology preparation process is quite involved, it is a one-time effort. This integrated ontology source thus become static and constant for all downstream ontology reuse applications.

Ontology Reuse Process Figure 1 shows how our system extracts an appropriate sub-domain from a larger, integrated source ontology by executing concept selection, relation retrieval, and constraint discovery. Since any ontology can be viewed as a conceptual graph, our algorithm is implemented to find nodes, edges, and specific constraints in graphs.

(1) Concept selection: We have implemented the concept-selection process as two concept-recognition procedures (which could be executed in parallel) and a concept-disambiguation procedure. In particular, the two recognition procedures involve concept-name matching and concept-value matching. Concept-name matching associates content in NL documents with concept names in source ontologies. For example, this procedure matches the word "capital" in the sentence "Afghanistan's capital is Kabul and its population is 17.7 million." to the ontology concepts CAPITAL-CITY and FINANCIAL-CAPITAL. The system hence associates both of the concepts as candidate concepts in the target domain. Concept-value matching associates content in NL documents with concept recognizers that have been pre-integrated during the source-ontology-preparation stage. For example, in the same sentence above, this procedure matches the word "Kabul" with the concept CAPITAL-CITY.

[6] http://wordnet.princeton.edu/

The concept disambiguation procedure follows the previous two recognition procedures. Since the source ontology contains thousands of concepts whereas the target domain may only contain dozens, we often encounter considerable ambiguity on selected concept candidates. In the previous example the system recognizes both CAPITAL-CITY and FINANCIAL-CAPITAL as matched for a common data instance. Concept disambiguation solves this type of problem. In our example, the system knows that CAPITAL-CITY and FINANCIAL-CAPITAL cannot both be valid candidates because the word "capital" should have one and only one meaning in the sentence. At the same time, since the concept-value procedure recognizes another instance "Kabul" of CAPITAL-CITY, but no more instances of FINANCIAL-CAPITAL, the system accepts CAPITAL-CITY and eliminates FINANCIAL-CAPITAL. We have implemented four concept disambiguation rules [5].

(2) Relation retrieval: The crux of the relation retrieval process is about finding appropriate edges between two concept nodes. An obvious resolution is to find all possible paths between any two candidate concepts. It would be easier for users to reject inapplicable edges rather than to add new relations. But this resolution has a serious performance difficulty. To find all paths between two nodes in a graph is NP-complete.[7] Hence we must seek an alternative resolution.

Different paths in an ontology graph refer to relations with different meanings. From our studies we have found that in general a shorter path represents a closer or more popular relationship between two concepts. On the contrary, an extra-long path often means a very uncommon relation between the two concepts within the domain. Hence it is reasonable to set a threshold length to reduce the search space and thus the complexity of the edge-searching algorithm.

In the implementation, we adapted the well-known Dijkstra's algorithm. Although the original algorithm computes only the shortest path, it can be easily extended by repeatedly computing the next shortest path until a threshold length is reached. Since Dijkstra's algorithm has polynomial time complexity and the threshold length is fixed and finite, the time complexity of this updated algorithm is also polynomial.

After this edge-searching procedure, the system perform a subgraph-detection procedure to finalize the target domain. Quite often, the edge-searching procedure results in multiple unconnected subgraphs. Normally, two separate subgraphs represent two independent domains. We simply assume that the largest subgraph contains the majority of concepts of interest, and thus the system keeps only the largest generated subgraph. Coincidentally, by rejecting concepts that are not in the selected subgraph, we further improve accuracy of domain recognition.

(3) Constraint Discovery. There are numerous types of constraints applied in ontologies; another paper would be necessary to enumerate them all and

[7] Finding the longest path between two graph nodes is a well-known NP-complete problem [9]. If we could solve finding all paths in polynomial time, by sorting the results in polynomial time, finding the longest path could also be solved in polynomial time—a contradiction.

to study the methods for reusing them. For our purpose in demonstrating automated ontology reuse, we limited our study to the discovery of cardinality constraints. Unlike many other constraints in ontologies, cardinality constraints contain quantitative scales, which make the automatic discovery process become interesting.

We have implemented a cross-counting algorithm to discover cardinality constraints from NL documents.[8] Each cardinality constraint consists of a pair with a minimum number and a maximum number $[min : max]$. The cross-counting algorithm counts the instantiated numbers of paired concepts, from which the system can decide these minimum and maximum numbers. For example, suppose that in document $D1$ concept A is instantiated by $a1$, and there are no instantiations for concept B in the same document. In another document $D2$, however, concept A is instantiated by the same $a1$ and concept B is instantiated by $b2$. With these two documents, we can determine that the minimum cardinality constraint of concept A to the relation AB is 0 because for an instance $a1$ of A, it may not always have an instance of B appearing at the same time. The details of this algorithm are presented elsewhere [5].

Ontology Refinement After finding concepts, relations, and constraints, composing them together into an ontology is straightforward. The result probably will not precisely describe the target domain. There are four basic human operations for revising the components in an automatically generated ontology: (1) remove the unexpected, (2) rename the inappropriate, (3) modify the incorrect, and (4) add the missing. In general, a preferred ontology reuse procedure will produce outputs requiring less revision operations on (3) and (4), especially the latter. It is in general much easier for users to reject unexpected components than to add something totally new into an ontology by themselves. Based on this refinement perspective, our ontology reuse system preserves as much useful information as possible, minimizing the need for addition by users.

4 Experiments and Discussions

This section describes a series of experiments with our ontology reuse system. We cast our discussion in five lessons that we believe are pertinent to future automated ontology reuse studies.

Lesson 1. Ontology coverage is best specified by the leaf concepts.

For ontology reuse, the coverage of an ontology is the reusable domain described by an ontology. Users for ontology reuse would be justified in believing that we can straightforwardly determine the coverage of an ontology by its root definition. For example, when the root concept of an ontology is *Book*, this ontology should cover the domain of books; when the root concept is Finanical Report, this ontology must cover the domain of financial reports. Since the root

[8] The original μK ontology does not contain information about cardinality constraints.

of the μK ontology is ALL (which means everything), as naïve users we began our study believing that we could reuse the μK ontology to describe arbitrary domains.

Our initial experiments produced disappointing output. Usually we either got no result or the composed ontologies were outside the expected domains. Careful study of the results located the problem: the real coverage of an ontology is not determined by its root definition. Although theoretically the root definition of an ontology should be an abstract characterization of the entire domain, often ontology developers do not properly circumscribe the domain and thus a significant portion of the domain is often not reusable. Instead, the real reusable domain of an ontology (i.e. the real coverage of an ontology) is primarily determined by the union of its leaf-level concepts, a subset of the root-specified domain. For example, if a *NATION* ontology contains leaf-level concepts like *USA*, *Russia*, *China*, *Australia*, etc., but lacks *Montenegro*, the concept *Montenegro* is not reusable with respect to this ontology. This observation is fairly simple though critical for ontology reuse research; interestingly, previous ontology reuse publications miss this point.

Lesson 2. Extend ontology coverage with lexicons and data recognizers.

To improve the degree of reusability of existing ontologies, we want to boost the coverage of an ontology so that it is closer to its root definition. We refer to this as the "applicable coverage" of an ontology, where the term "applicable" means the new concepts can be evaluated by an ontology reuse program.

To boost the applicable coverage of our source ontology during the source-ontology preparation stage, we associated lexicons and data recognizers with the leaf-level concepts. We have named the result "instance recognition semantics", or formal specifications that identify instances of a concept C in ordinary text [6]. These are essential to automating ontology reuse.

We further populate the ontology with some upper-level ontology concepts. For example, prior to June 3, 2006 Montenegro was not an independent nation, so the original μK ontology did not have a leaf concept *Montenegro* under *NATION*. This portion of the ontology becomes non-reusable for many situations involving Montenegro after June 3, 2006. It is a very complicated issue to get permissions and then properly modify an ontology that is created by somebody else. For the purpose of automated reuse, however, we developed a simple and effective (though imperfect) alternative. We simply bind a lexicon set to the non-leaf concept *NATION*, thus adding the name of Montenegro into the lexicon after June 3, 2006. Although we still have not formally specified Montenegro as a country in the ontology, we have rendered the original source ontology reusable for situations involving the new country Montenegro. In the new generated ontology, instead of a specific concept *Montenegro* as an independent nation, we can correctly generate an upper-level concept—*NATION*, and thus all the properties of *NATION* become applicable in this new generated domain ontology. With such a technique, we artificially boost the applicable coverage of the source ontology.

In our experiments we augmented lexicons and data recognizers for leaf-level concepts in the μK ontology and their superclasses up to 2 levels above (on

average). The union of these augmented concepts and their relations composes the applicable coverage of the source ontology in our experiments.

Lesson 3. For known target domains, ontology reuse is already possible and even valuable.

After having prepared the source ontology, we started our real experiments. Based on Lesson 1, we decided to focus our experiments on several selected domains rather than on arbitrary domains. We want human inspection to assure that the projecting domains have significant overlap with the applicable coverage of our source ontology. In particular, we chose experiments in three narrow domains: car advertisements, apartment rentals, and nation descriptions. This paper only briefly summarizes our results; see [5] for details.

First we list some basic settings and statistics of our experiments. Each of the three target domains contains a dozen to twenty concepts. For each domain, we feed four to seven cleaned sample Web pages (NL documents) to the ontology reuse system. The source ontology has been pre-integrated and augmented by its applicable coverage. In order to evaluate the performance of our outputs, we had human experts separately create ontologies for each target domain. We adopted the human-created ontologies as a gold standard to which the automatically generated ontologies were compared for precision and recall.

In general, we obtained low precision results. In the three target domains, the best precision was 48% for concept generation, 14% for relation generation, and 10% for cardinality constraint generation. The news is not all bad. Low precision implies the need for more rejections of corresponding components within a generated ontology. For humans, as mentioned earlier, rejecting inappropriate ontology components is much easier than adding new ontology ones. Hence our strategy is to favor greater recall values (i.e. less addition) over greater precision values (i.e. less rejection).

We updated the traditional recall calculation equation as follows:

```
updated recall = # correctly-reused / # existing-in-source
```

where the numerator is the number of component types (i.e. either concept, relationship, or constraint) correctly reused in a generated ontology; the denominator is the number of component types contained in input sources (both from NL documents and source ontologies). We use this formula because not everything defined in the human-created ontology is also identifiable by the inputs. For example, human experts have defined a concept *FEATURE* in the car-ads ontology, a concept missing from the source μK ontology. Hence it is impossible for a system to reuse an non-pre-existing concept. To be more equitable, our recall calculation must eliminate this type of error.

With the new formula, in the three testing domains our worst recall values were 83% (concept generation), 50% (relation generation), and 50% (cardinality constraint generation). All the best recall values were close or equal to 100%. Our ontology reuse system performs quite well even though it still is a prototype. The recall values show that we may reduce at least half of the human effort in

ontology construction through ontology reuse when a target ontology is properly contained in the applicable coverage of the source ontology. Considering the expense of training professional ontologists and the time they need to build and tune ontologies, 50% already represents substantial savings. There are many ways to further improve the performance of the system. Already, though, our experiments demonstrate that ontology reuse is no longer "far from an automated process" [13].

Lesson 4. Ontology modularization facilitates automated ontology reuse.

During our experiments, another metric studied was running time. In general the system took about 1000 seconds to resolve all the ontology components with respect to about 50 to 100 candidate concepts on a Pentium 800 MHz single processor machine. This execution time is rather short compared to the time required for manually creating an ontology of the same scale. Our benchmark showed that almost 90% of execution time was spent on the relation retrieval process. Though we may further improve this time performance by optimizing our implementation, the problem lies mainly in the magnitude of the source ontology (over 5000 concepts and over 70000 relationships to explore).

Reducing the execution time of relation retrieval should be possible by using modular ontologies rather than a single large-scale one. Modular ontologies are usually small and designed to be self-contained. An ontology module is self-contained if all of its defined concepts are specified in terms of other concepts in the module, and do not reference any other concepts outside the module. As soon as several major concepts in a module are selected as candidate concepts, an ontology reuse system may decide to directly reuse the entire module rather than perform a costly relation retrieval algorithm. Hence the execution time for relation retrieval can be significantly reduced.

To pursue this issue, we manually pruned several comparatively independent clusters of ontology components from our source ontology and used them as individual modules. Since these clusters originated from a previously unified ontology, we did not need to further integrate them. The same experiments were re-run with these multiple "modular" ontologies. On average the system took less than 300 seconds—saving more than 70% of run time—to resolve all the ontology components for about 50 to 100 candidate concepts. Because these pruned clusters were not true, self-contained modular ontologies, the performance in terms of precision and recall decreased in this experiment. Saving execution time by replacing a large unified ontology with multiple small modular ontologies is thus a convincing strategy. By using really well-designed modular ontologies, our ontology reuse system achieves both higher precision and recall values, as well as faster run-time performance.

Lesson 5. Sample documents may help us mine "latent" knowledge from texts.

We also carefully studied our low-precision experimental results. Many reused concepts and relations were beyond the scope of the expert-created ontologies. Yet they were not all meaningless or useless. On the contrary, we found that

useful information—latent in the document but beyond the topic directly at hand—could be gleaned from the results.

For example, we have applied our tool to process some U.S. Department of Energy (DOE) abstracts. The expert who created a reference ontology was only interested in the generic information about these abstracts, such as the theme of a document, the number of figures and tables, etc. But our ontology reuse tool found much more useful information. For instance, in one sample abstract it generated some concepts and relations indicating that the crude oil price dropped in the year 1986. Although this was not what the human expert originally expected and it was outside the expert-specified domain of interest, we could not deny that this type of information could be very valuable.

Such latent information is not really what people cannot find. But they are easily overlooked by human readers, especially when reading through many such documents. Especially within the business domain, people want to mine this type of latent information from numerous financial documents and business news. We believe that the automated ontology reuse mechanism may provide the business community an alternative solution for seeking valuable latent information.

5 Conclusion

We have presented an automated ontology reuse approach. Although we only applied our system to reuse the μK ontology, our methodology supports automated ontology reuse in general. Informed by our experiments on real-world examples, we have summarized five lessons that are constructive for future exploration of ontology reuse studies. In essence, we conclude that ontology reuse is no longer "far from an automated process" [13].

In the meantime, a few critical problems remain to be solved. One is to automatically decide whether a target domain is within the reusable coverage of an integrated source ontology. If the majority of a target domain lies outside the source ontology, ontology reuse becomes nothing but extra overhead. Also, we need to experiment with applying modular ontologies for ontology reuse. Until now, the research of modular ontologies is still at the stage of theoretical analysis. We need practical study cases to push this research field forward. The study of instance recognition semantics should be paired with modular ontology research to improve the reusability of modular ontologies. Last but not least, mining latent information through ontology reuse is an interesting research topic. More exploration on this topic may bring many benefits to users, especially in the business domain.

So far there are few published studies on automated ontology reuse research. We hope that our results draw more attention to this field and facilitate wider public adoption of the Semantic Web.

References

1. H. Alani, S. Harris, and B. O'Neil. Ontology winnowing: A case study on the AKT reference ontology. In *Proc. Int'l Conference on Intelligent Agents, Web Technology*

and Internet Commerce (IAWTIC'2005), Vienna, Austria, Nov. 2005.
2. J. Bao, D. Caraagea, and V. Honavar. Modular ontology – a formal investigation of semantics and expressivity. In *Proc. First Asian Semantic Web Conference (ASWC 2006)*, Beijing, China, Sept. 2006. (in press).
3. E. Bontas, M. Mochol, and R. Tolksdorf. Case studies on ontology reuse. In *Proc. 5th Int'l Conf. on Knowledge Management (I-Know05)*, Graz, Austria, 2005.
4. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. Contextualizing ontologies. *Journal of Web Semantics*, 1(4):325–343, Oct. 2004.
5. Y. Ding. Semi-automatic generation of resilient data-extraction ontologies. Master's thesis, Brigham Young University, Provo, Utah, June 2003.
6. Y. Ding, D. Embley, and S. Liddle. Automatic creation and simplified querying of semantic web content: An approach based on information-extraction ontologies. In *Proceedings of the first Asian Semantic Web Conference (ASWC 2006) LNCS 4185*, pages 400–414, Beijing, China, Sept. 2006.
7. Y. Ding and D. Fensel. Ontology library systems: The key for successful ontology reuse. In *Proc. First Semantic Web Working Symposium (SWWS'01)*, Stanford, CA, July 2001.
8. B. Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularizing OWL ontologies. In *Proc. KCAP-2005 Workshop on Ontology Management*, Banff, Canada, Oct. 2005.
9. D. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, MA, 1997.
10. K. Mahesh. Ontology development for machine translation: Ideology and methodology. Technical Report MCCS-96-292, Computer Research Laboratory, New Mexico State Univeristy, 1996.
11. N. Noy and M. Musen. Specifying ontology views by traversal. In *Proc. Third International Semantic Web Conference (ISWC 2004)*, pages 713–725, Hiroshima, Japan, Nov. 2004.
12. H. Stuckenschmidt and M. Klein. Structure-based partitioning of large class hierarchies. In *Proc. Third International Semantic Web Conference (ISWC 2004)*, pages 289–303, Hiroshima, Japan, Nov. 2004.
13. M. Uschold, M. Healy, K. Williamson, P. Clark, and S. Woods. Ontology reuse and application. In *Proc. International Conference on Formal Ontology and Information Systems (FOIS'98)*, pages 179–192, Trento, Italy, June 1998.
14. M. Uschold and M. King. Towards a methodology for building ontologies. In *Proc. Workshop on Basic Ontological Issues in Knowledge Sharing in conjunction with IJCAI-95*, Montreal, Canada, 1995.