ADDING SUPPORT FOR DYNAMIC ONTOLOGIES TO EXISTING KNOWLEDGE BASES

Upmanyu Misra, Zhengxiang Pan, Jeff Heflin
Department of Computer Science and Engineering, Lehigh University,
19 Memorial Dr West, Bethlehem, PA 18015, USA

Keywords: Ontology, Ontology Versioning, OWL, Semantic Web technology, Intelligent Agent

Abstract: An ontology version needs be created when changes are to be made in an ontology while keeping the basic structure of the ontology more or less intact. It has been shown that an Ontology Perspective theory can be applied on a set of ontology versions. In this paper, we present a Virtual Perspective Interface (VPI) based on this theory that ensures that old data is still accessible through ontology modifications and can be accessed using new ontologies, in addition to the older ontologies which may still be in use by legacy applications. We begin by presenting the problems that are to be dealt with when such an infrastructure needs be created. Then we present possible solutions that may be used to tackle such problems. Finally, we provide an analysis of these solutions to support the one that we have implemented.

1 INTRODUCTION

Many discrepancies may be caused due to incompatible changes between ontologies. These changes may happen when various versions of an ontology are created where new concepts are added, for example when an abstract super class for two classes is added, or a class is moved, renamed etc. It has been shown that an Ontology Perspective theory (Heflin and Pan, 2004) can be applied on a set of ontology versions. In this paper, we present a Virtual Perspective Interface (VPI) based on this theory that ensures that old data is still accessible through ontology modifications and can be accessed using new ontologies, in addition to the older ontologies which may still be in use by legacy applications. VPI obviates the need for wholesale translation of existing data every time an ontology is changed. The focus of VPI is on prospective use (i.e., using newer versions of some ontologies to access knowledge represented in previous versions of those ontologies) versus retrospective use (i.e., using older versions of some ontologies to access knowledge represented in newer versions of those ontologies).

We begin, in section 2, by providing relevant background for the technologies used. We also define the problem we are addressing in this paper and give some motivation for the need of a system that solves the problem. In section 3, we present the architecture for such a system. We produce the VPI modules and the implementation details. This is followed by an analysis of the system in section 4. In section 5, we present snapshots of our implemented system and some samples. Finally, in section 6, we conclude with an overview of our system and future work.

2 BACKGROUND

In this section, we will provide a brief description of related terms that will be frequently used throughout this paper and some discussion of the research that our paper builds up on.

2.1 Ontologies for Information Systems

Ontology is a central issue for the development of Internet commerce systems. The main barrier to electronic commerce lies in the need for applications to meaningfully share information, not in the reliability or security of the Internet. This is because of the variety of enterprise and e-commerce systems

Misra U., Pan Z. and Heflin J. (2005).
ADDING SUPPORT FOR DYNAMIC ONTOLOGIES TO EXISTING KNOWLEDGE BASES.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 97-104
DOI: 10.5220/0002529700970104

deployed by businesses and the way these systems are variously configured and used. Although it is useful to strive for the adoption of a single common domain-specific standard for content and transactions, this is often difficult to achieve, particularly in cross-industry initiatives, where companies co-operate and compete with one another. In this regards, there has been previous work to develop a core ontology which may be used as a base to develop domain specific ontologies (Hunter, 2003).

For over a decade, knowledge representation researchers have studied the use of ontologies for sharing and reusing knowledge (Gruber, 1993; Guarino, 1998; Noy and Hafner, 1997). Although there is some disagreement as to what comprises an ontology, most ontologies include a taxonomy of terms (e.g., a Car is a Vehicle), and many ontology languages allow additional definitions using some form of a logic. Guarino (1998) has defined an ontology as "a logical theory that accounts for the intended meaning of a formal vocabulary." A common feature in ontology languages is the ability to extend preexisting ontologies. Thus, users can gain the interoperability benefits of sharing terminology where possible, but can also customize ontologies to include domain specific information.

Research on the development of the Semantic Web (Berners-Lee et al., 2001) has led to progress in the design and use of distributed ontologies. Two languages have been developed for representing describing and semantics on the Web. The Resource Description Framework (RDF) is provides a graph-base data model, in which every resource is identified by a Unified Resource Identifier (URI). For syntactic convenience a URI can be abbreviated using namespace prefixes which can be defined in each document. An arc in the graph can be viewed as a triple consisting of subject, predicate and object. For example, <subject, subClassOf, object> means that subject is a subclass of object. The Ontology Web Language (OWL) extends RDF to provide a richer set of modeling constructs that allow the semantics of data to be more precisely defined. These languages can be used to describe the semantic for enterprise information systems.

However, it is usually impossible to establish, a priori, rules (technical or procedural) governing participation in an electronic marketplace. In the fast paced scenario of e-commerce and m-commerce, an enterprise must be able to adapt its information systems quickly. We, in this paper, are focussing on ontology changes. Hence the ultimate goal is the development of reusable, dynamic ontologies (Heflin and Hendler, 2000) that can be applied across multiple disciplines. This calls for a comprehensive framework to formulate and maintain ontology versions. Ontology Versioning Theory (Heflin and Pan, 2004; Heflin 2001), in section 2.2, presents such a framework.

2.2 Ontology Versioning

Once a database schema changes one faces the problem of managing the data using its different versions in a consistent and economical fashion. In this context, database schema versioning (Roddick, 1995) is similar to the ontology versioning. Roddick pointed out two ways database schema can be viewed as, prospective (viewing data from the point of view of a newer ontology) and retrospective (viewing data from the point of view of an older ontology). Klein and Fensel (2001) were the first to compare ontology versioning to database schema versioning. They proposed that both prospective use and retrospective use of data should be considered in ontologies as well. However, Klein and Fensel do not describe a formal semantics.

Stuckenschmidt and Klein, (2003) provide a formal definition for modular ontologies and consider the impact of change in it. However, their approach involves physical inclusion of extended ontologies and requires that changes be propagated through the network. This approach is unlikely to scale in large, distributed systems. Furthermore, they do not allow for resources to be reasoned with using different perspectives, as is described here.

Heflin, (2001) has suggested that there should not be a universal model of all the resources and ontologies on the Web. In fact, it is extremely unlikely that one could even exist. Instead, we must allow for different viewpoints and contexts, which are supported by different ontologies. Ontology Perspective Theory from Heflin (2001) defines perspectives which then allow the same set of resources to be viewed from different contexts, using different assumptions and background information.

Heflin and Pan (2004) builds on this and presents a model theoretic description of ontology perspectives. Each perspective is based on an ontology, called the basis ontology or base of the perspective. By providing a set of terms and a standard set of axioms, an ontology provides a shared context. Thus, resources that commit to the same ontology have implicitly agreed to share a context. We also want to maximize integration by including resources that commit to different ontologies. This includes resources that commit to ancestor ontologies and resources that commit to earlier versions of ontologies that the current ontology is backward compatible with. Heflin defines that an ontology version is backward

compatible with a prior version if it contains all of the terms from the earlier ontology and the terms are supposed to mean the same thing in both ontologies. See the paper for details regarding entailment in the theory.

We use Web Ontology Language (OWL) to represent knowledge bases. But OWL has little to no semantics for versioning properties yet. Ontology Perspective Theory allows multiple views of the same data where each view is based on an ontology. It also provides an initial semantics for versioned ontologies, in particular, semantics for backward-compatibility.

Prospective use of data is a highly desirable concept and backward compatibility facilitates this. Hence its use will be an incentive for ontology engineers. However in the present OWL world we can't state that backward compatible versions are more common than ones that are not. OWL is too new and there aren't many, if any, new versions of existing ontologies. However, we argue that, if the versions are truly incompatible then it does not make sense to reuse data. For example, if a new version significantly alters the modeling of knowledge in the previous version, it may not be useful to construct a link between the two versions. Moreover, backward compatibility could allow 'deprecation' of terms between two versions and hence is more flexible than it might initially seem. Say there are two incompatible versions of an ontology: $O_{v1}$and O_{v2}. We can have a $O_{v1.5}$ which is compatible with O_{v1} and deprecate all the terms that will disappear in O_{v2}. Then O_{v2} can delete those deprecated terms while being compatible with $O_{v1.5}$. Thus deprecation facilitates backward compatibility without hindering the validity of a previous version that might be in use by some other application.

3 VPI

As an ontology evolves, multiple versions are created. These versions coexist in the containing knowledge base. Most legacy applications will want to keep the snapshots of these versions at all time and query for different data with respect to different versions. Additionally, many Enterprise Systems are inherently distributed and dependent.

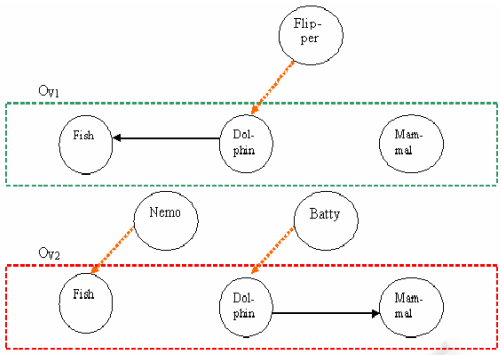

Figure 1: Simple Example of Ontology versioning

This makes it undesirable to discard the data contained in any of the older versions. In practise, there will be many complex cases that will result in Ontology versioning. One such case may be where a subclass is deleted. Fig. 1 denotes a part of ontology concerning Fish genealogy. O_{V1} wrongly conveys that the class 'dolphin' is a subclass of class 'fish' whereas O_{V2} rectifies this and reassigns the 'dolphin' class to be a subclass of class 'mammal'. Additionally, Flipper and Batty are instances of class 'dolphin' in O_{V1} and O_{V2} respectively. Therefore, according to O_{V1}, Flipper is a dolphin and hence a fish. Whereas, Batty is a type of dolphin, and hence is a mammal. This is also a case demonstrating the use of backward-compatibility. It must be emphasized here that the intended meaning (and not only the name) of a class must remain the same in subsequent versions. For example, the dolphin, in version 2, must remain to be the living creature that is commonly known and not a different entity.

This requires us to come up with a system where such discrepancies can be removed while retaining as much knowledge as possible. VPI provides a simple yet effective method to achieve these goals. It is designed such that it sits on top of the underlying model of knowledge bases and acts as an agent mediating between the querying user and the knowledge bases. For example, a query for 'Fish' from the perspective of version 2 must produce Nemo. The same query from the perspective of version 1 will give us Flipper. A query for 'Mammal' from the perspective of version 2 will give us both Batty and Flipper. The Flipper solution comes from the inference using version 1. Without using perspectives, our answer would have been Batty only.

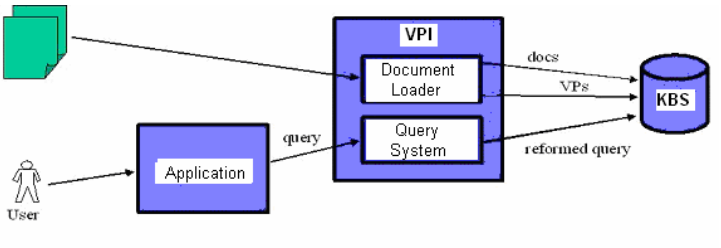

Figure 2: High Level Design of VPI

3.1 Architecture

VPI acts as the middle layer between the user and any OWL knowledge base system. It facilitates the knowledge base to perform perspective based entailment without modifying any internal knowledge in the system. It creates a virtual OWL Ontology, the details of which are provided later, that results in perspective entailment using only OWL entailment. This is shown in Fig. 2. There are two main modules of the VPI.

Document Loader: The document loader takes an ontology and computes the virtual perspective ontology for it. This will be OWL file with the same local names as the other ontology, but will contain mappings to ontologies with which the ontology is backward-compatible. Both the original ontology and the virtual perspective ontology will be stored in the underlying knowledge base.

Query System: All queries to the knowledge base pass through the VPI. An application may either select a default perspective or the user may specify a perspective for each query. The VPI translates the query to the terms of the corresponding virtual perspective ontology, and issues the query to the underlying knowledge base. It then returns the results to the client application.

For all the following discussion, we will use the following convention for naming ontologies,
Version X of any ontology is written as O_{VX}. Hence version 4 of an ontology will be O_{V4}. To denote a Virtual Perspective we will use O_{VP}. The class C of a version, say 3, will be denoted as O_{V3}:C.

Fig. 3 is a graphical example of a naïve solution to our problem. In this case there are two ontologies, O_{V1} and its latest version O_{V2}. It creates mappings such that all O_{V1} and O_{V2} classes are made subclasses of their corresponding VP classes.

Note: VP is in perspective of O_{V2} here. In the new version, it was established that Dolphin is a subset of Mammal rather than a subset of Fish. The VP was constructed using O_{V2} as the perspective. The orientation of the class hierarchy is such that X, which is a O_{V1}:dolphin, is labeled as both, a O_{VP}:fish (due to version 1) and a O_{VP}:mammal (due to version 2).

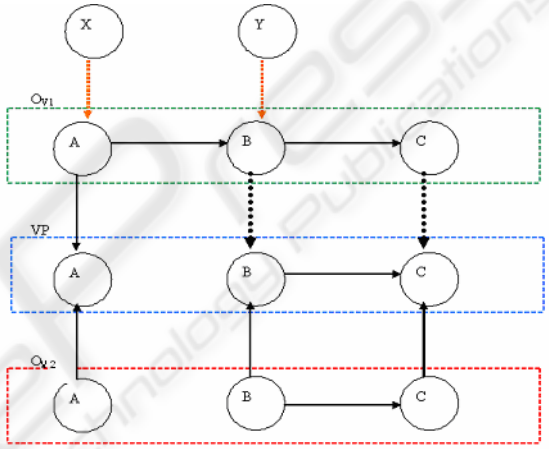

Figure 3: Erroneous Virtual Perspective

It may be argued that such an outcome is undesirable since the primary reason for versioning the ontology might have been to substitute the knowledge that dolphins are mammals with the earlier version of them being fish. This is the approach taken by ontology perspective theory. The axioms of the new ontology are used in place of the old. We would still like to keep the knowledge for all O_{V1}:fish that are not dolphin, such as Y. Hence the system should be equipped such that it can make intelligent inferences in such situations.

We start with O_{V1} where A is a subclass of B which is a subclass of C. In its version O_{V2}, A no longer remains a subclass of B. The O_{VP} is built with O_{V2} as the perspective.
In this case it is not possible to make any more links of the version classes with the O_{VP} classes as in Fig. 3 (the dotted links are faulty). This is because as soon as we make a link, we will be confronted with the problem that is mentioned above. For example, assume a “subclassOf” relationship from O_{V1}: B to O_{VP}:B. Using the VPI inferencing, X will be type O_{V2}:B and hence also of type O_{V2}:C. Using perspective of O_{V2}, we should not be able to say that

any instance of O_{V2}:A is necessarily an instance of O_{V2}:B.

3.2 Solution

We looked at some possible techniques for the solution. Later we briefly describe and compare them in terms of various attributes such as their scalability, soundness and completeness. Here we explain the technique that we finally accepted and implemented as the solution.

Before that we will delineate the some more notations used throughout the paper. For denoting triples we use <Subject, Predicate, Object>. We will use colons to indicate namespaces whenever relevant. Hence, considering ontology in fig. 1, O_{V1}:A implies a class A in ontology version 1. Hence B is a sub class of A can be written as,

<B, subClassOf, A>

and I is a type of a Class C can be written as,

<I, typeOf, C>

Note: "type of" is a synonym of "instance of".

This method makes use of a separate "type graph" that is formulated for every ontology. Each ontology will have two virtual ontology. The virtual perspective ontology will be identical to the original except for a different namespace. The type graph ontology contains copy of each class and type values (instances) for all classes without any relationships (property). Each version will have its own type graph. The linking between each version will be done using their type graphs. Let's consider the same example as in fig. 3.

For type graphs, we will be using the following convention, the type graph of version k will be written as O_T^k, and O_T will be any type graph in general. The pseudo code for constructing a virtual perspective graph and a type graph is as follows,

// Copy classes and properties into the virtual ontologies
For each triple <O_n:C type rdfs:Class> in O_n
Add <O_T:C type rdfs:Class> to O_T
Add <O_V:C type rdfs:Class> to O_V
Add <O_T:C subClassOf O_V:C> to O_V
For each triple <O_n:P type rdf:Property> in O_n
Add <O_V:P type rdf:Property > to O_T
Add <O_V:P type rdf:Property > to O_V
Add <O_T:P subPropertyOf O_V:P> to O_V
// Copy axioms into virtual perspective ontology
For each triple <O_n:C_1 subClassOf O_n:C_2> in O_n
Add <O_V:C_1 subClassOf O_V:C_2> to O_V
For each triple <O_n:C_1 subPropertyOf O_n:C_2> in O_n
Add <O_V:C_1 subPropertyOf O_V:C_2> to O_V
//Link corresponding instances to type graph
For each triple <O_n:I type O_n:C>
Add <O_T:I type O_T:C> to O_T

This gives us a representation of each ontology version with type graphs and virtual perspective graphs linked together. Now we present three candidate solutions for linking each ontology version to its backward compatible version. Let B be the set of backward compatible versions for each of the ontology.

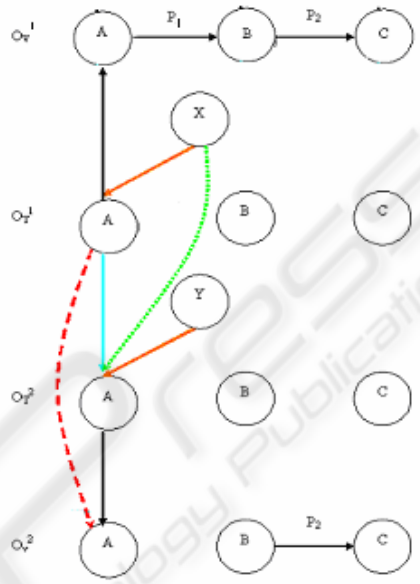

Figure 4: Various Type Graph implementations

Instance to type graph mapping *(green dotted lines in fig. 4)***:** All the instances in a type graph version are mapped onto corresponding classes of all the subsequent backward-compatible versions of that type graph.

Type graph to virtual perspective mapping *(red dashed lines in fig. 4)***:** All classes and properties in the type graph are mapped to their sisters in the virtual perspective graphs of all ontologies that are backward compatible with the current ontology.

Type graph to type graph mapping *(blue line in fig. 4)***:** To link type graphs such that we can successfully access all relevant ontologies, we add the following pseudo code to the one that we previously presented,

//Find the ontology $O_b \in$ B, such that O_b immediately //precedes O
For each triple <O_b:C type rdfs:Class> in O_b
Add < O_T^b:C subClassOf O_T> to O_T
For each triple <O_b:C type rdfs:Property> in O_b
Add < O_T^g:C subPropertyOf O_T> to O_T

Fig. 4 shows an example of the above discussed solution applied on a simple test case. An ontology O_{V1} is shown in the first set of the figure. For the

next version O_{V2} of the same ontology, property P_1 is deleted. Also an instance Y of type A is added. Their corresponding type graphs are also shown. The link between the type graphs is shown by the blue solid line. In the next section, we will compare these three alternatives.

4 ANALYSIS

The three variations of the algorithm can be evaluated with respect to different criteria. First, the algorithms must be sound with respect to ontology perspective theory, meaning that any result deduced by the algorithm must be entailed by the theory. Second, the algorithms must be complete with respect to the theory, meaning that any formula entailed by the theory must be inferred b the algorithm. Third, we can distinguish the scalability of the algorithms, based on the number of triples that they require.

All of the algorithms are sound and complete for ontology perspective theory, assuming that the knowledge base system is sound and complete for OWL. However, due to limited space we omit the proof of this.

However, all three algorithms differ in terms of scalablity. In each method, triples have to be replicated many times. Given the enormous size of ontologies to deal with, scalability is a major priority.

Given below are a few variables that will be used throughout the analysis,

No. of versions of ontologies: n

No. of instances per class: I

In the following cases we compare different techniques mentioned earlier in terms of the total number of extra triples required to be created when each technique is applied. Consider that the process of creating links between classes has reached upto the k^{th} version.

For ease of analysis we assume that each O_k is backward compatible with its previous version, each ontology contains m classes such that $<C_{m-1}$, subClassOf, $C_m>$ (for k > 1), and, Version O_k removes the $(k-1)^{th}$ "subClassOf" link.

Instance to type graph mapping: Since every instance must be linked to the type graph of every subsequent version, the number of additional triples is I (*m*)(n)(n-1)/2. This is because linking the instances of any ontology to one subsequent ontology requires I (*m*) triples, and this must be done $\sum_{i=1}^{n-1} i$ = (n)(n-1)/2 times.

Type graph to virtual perspective mapping: Every class in the type graph must be linked to every subsequent virtual perspective graph, which requires $\sum_{i=1}^{n-1} i$ = (n)(n-1)/2 links per class. Thus this approach requires (*m*)(n)(n-1)/2extra triples.

Type graph to type graph mapping: Since each type graph only establishes links to a single preceding type graph, and every class must have a link, the number of triples will be (n-1)*m*.

Clearly, the instance to type graph mapping approach is the least scalable. Furthermore, as long as $n > 2$, then the type graph to type graph mapping requires fewer triples than type graph to virtual perspective mapping. For example, for n=10, type graph to virtual perspective mapping requires 5 times as many additional triples as type graph to type graph. We decided that this improved scalability more than offset the potential query time advantage of the type graph to virtual perspective mapping approach, and thus implemented type graph to type graph mapping.

Additionally, the type graph method is simple to comprehend, implement and present. Taking all these factors into consideration, the type graph method seems to be a befitting approach. Hence we implemented the VPI with type graph method.

5 VPI IMPLEMENTATION

We have implemented the VPI in Java. This implementation make use of Jena, a Java framework for semantic web applications. Jena provides a programmatic environment for RDF, RDFS and OWL, including a rule-based inference

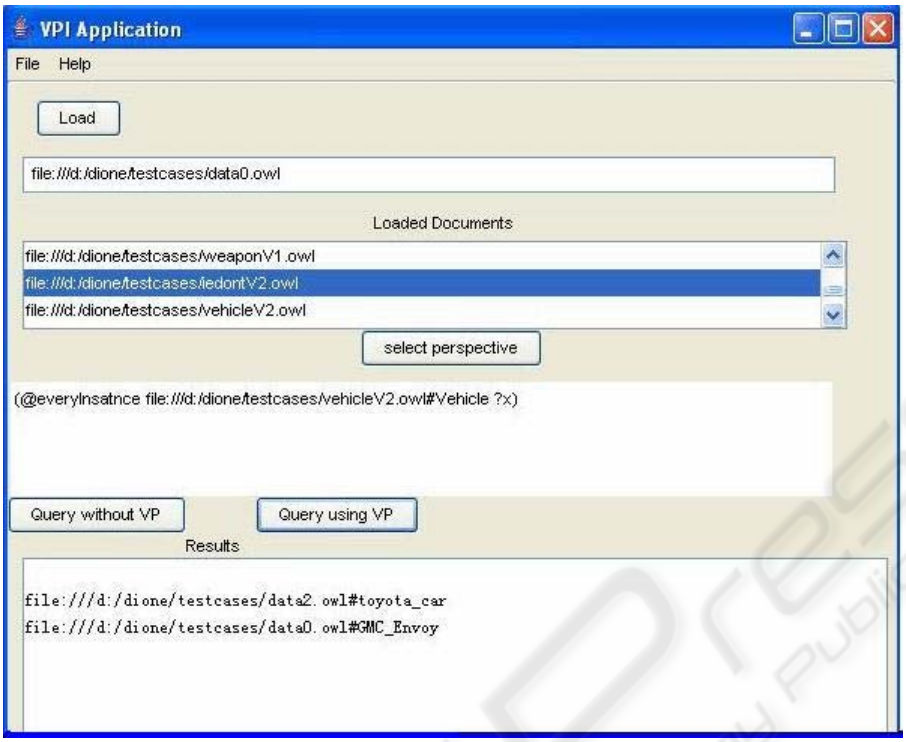

Figure 5: VPI User Interface

engine. The core of the VPI is a Java class that consists of two methods, loadDocument() and issueQuery(), that have the functionality already discussed in section 3.1.

In order to evaluate the VPI, we created a simple user interface shown in fig. 5. This interface allows the user to load in OWL documents one at a time and issue queries using different perspectives. The user selects a perspective from a list of loaded ontologies and types a KIF-like query into a text box. When the user presses the "Query using VP" button, the VPI is used to issue the query, and the answers are returned in the Results field. If the user presses the "Query without VP" button, the query is issued directly to the knowledge base, circumventing the VPI. Using these two methods, the user can see the advantages of ontology perspective theory over standard OWL entailment.

6 CONCLUSION

In this paper we discussed the problems that may arise when we deal with a group of ontology and its versions. It was argued that both discarding the older version and storing redundant or obsolete data are naïve ways to handle ontologies. The former results in loss of knowledge whereas the latter induces overheads on computation. We provided a simple, yet effective, solution to tackle the discrepancies that arise. We saw how several techniques qualifies as potentials solutions. We gave relevant analysis to sift out the most suitable solution. In future, we plan to enable handling of imports, i.e. support perspectives for ontologies that import other ontologies. We will also be looking at deploying and testing an OWL reasoner. This will facilitate the handling of versioning for richer ontologies, i.e. those that contain OWL axioms rather than just RDFS axioms. This will require an alternative to Jena and we are looking at HAWK and DLDB, developed at Lehigh. DLDB is a knowledge based systems developed for large semantic web applications and includes a description logic reasoner that implements a significant fragment of OWL referencing. Finally, an empirical evaluation comparing this approach with other alternatives will be desirable.

REFERENCES

Berners-Lee, T., Hendler, J., Lassila, O., 2001. The Semantic Web. In Scientific American.

Gruber, T., 1993. A Translation Approach to Portable Ontology Specs. In Knowledge Acquisition.

Guarino, N., 1998. Formal Ontology and Information Systems. In *Formal Ontology and Information Systems*. IOS Press.

Heflin, J., Hendler, J., 2000. Dynamic Ontologies on the Web. In *AAAI'00, 7th National Conference on Artificial Intelligence*. AAAI/MIT Press.

Heflin, J., 2001. Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment. PhD thesis, University of Maryland.

Heflin, J., Pan Z., 2004. A Model Theoretic Semantics for Ontology Versioning. In *ISWC'04 ,3rd International Semantic Web Conference*. Springer.

Hunter, J., 2003. Enhancing the Semantic Interoperability of Multimedia through a Core Ontology. In *ICEIS'03, 5th International Conference on Enterprise Information Systems.*

Klein, M., Fensel, D., 2001. Ontology Versioning for the Semantic Web. In *SWWS'01, 1st Semantic Web Working Symposium*.

Noy, N., Hafner, C., 1997. The State of the Art in Ontology Design. In AI Magazine.

Roddick, J., 1995. A Survey of Schema Versioning Issues for Database Systems. In Information and Software Technology.

Stuckenschmidt, H., Klein, M., 2003. Integrity and Change in Modern Ontologies. In *IJCAI'03*.