An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies

A. Maedche and B. Motik and L. Stojanovic and R. Studer and R. Volz
FZI Research Center for Information Technologies at the University of Karlsruhe
Haid-und-Neu Strasse 10-14
76131 Karlsruhe, Germany
{maedche,motik,stojanov,studer,volz}@fzi.de

ABSTRACT

The vision of the Semantic Web can only be realized through proliferation of well-known ontologies describing different domains. To enable interoperability in the Semantic Web, it will be necessary to break these ontologies down into smaller, well-focused units that may be reused. Currently, three problems arise in that scenario. Firstly, it is difficult to locate ontologies to be reused, thus leading to many ontologies modeling the same thing. Secondly, current tools do not provide means for reusing existing ontologies while building new ontologies. Finally, ontologies are rarely static, but are being adapted to changing requirements. Hence, an infrastructure for management of ontology changes, taking into account dependencies between ontologies is needed. In this paper we present such an infrastructure addressing the aforementioned problems.

Categories and Subject Descriptors

H.3.4 [**Software**]: Information Systems—*Distributed systems*; D.2.13 [**Software**]: Software Engineering—*Reuse models*

General Terms

Management

Keywords

ontology registry, ontology reuse, ontology evolution

1. INTRODUCTION

Nowadays Semantic Web applications are still in an early stage. Ontology development is difficult and time-consuming, and this is even worsened by the fact that ontologies are mostly created from scratch, resulting in the "Babel of Ontologies" problem. In general it can be said that there is a lack of methods and tools supporting and facilitating ontology reuse. Furthermore, processes for evolving distributed ontologies haven't yet been established, thus making propagation of changes to distributed ontologies impossible.

To better understand the problem of searching, reusing and evolving distributed ontologies in the Web, we consider the following B2B catalog integration scenario throughout the whole paper. Let's assume that a service provider A produces various sports utilities and that he wants to publish his catalog on-line. To enable other players in the market place to semantically process the catalog, he creates a sports ontology (SO) and bases his catalog on it. Let's assume that the service provider B specializes in production of bicycles and also wants to publish his catalog on-line. To achieve that, he will create a bicycle ontology (BO) for catalog description. In doing so, he should reuse as many definitions as possible from existing ontologies to speed up the engineering and to enable interoperability. However, two challenging problems arise:

- It is not clear how to locate ontologies that can be used as a basis for developing BO.
- Even if an appropriate ontology has been found, it is not clear how to reuse its definitions in BO.

After reusing SO as the basis for BO, a further problem arises when SO needs to be adapted due to change in business requirements. Thus, the fundamental question of how to evolve dependent ontologies arises. This problem is worsened by the fact that ontologies are distributed in the Web.

From this scenario we derive three important ontology infrastructure components: First, an *ontology registry* providing means for locating existing ontologies is required. Second, means for *reusing* distributed ontologies are required. Third, we consider methods supporting the consistent *evolution of distributed ontologies* as crucial for the success of the distributed system in the long run. This paper describes an integrated approach for establishing an infrastructure for searching, reusing and evolving distributed ontologies. The approach has been implemented within the KAON ontology management framework[1].

The rest of this paper is structured as follows. In section 2 we give an overview of our infrastructure and identify its key components. We elaborate these components further in sections 3, 4 and 5. In section 6 we discuss how the infrastructure described in this paper has been implemented. Before we conclude, in section 7 we present an overview of the related work.

2. INFRASTRUCTURE OVERVIEW

This section presents at a high-level overview of how our integrated approach works. The B2B catalog integration scenario is presented in more detail in Figure 1. We envisage the Semantic Web being realized using many ontology servers (nodes) that cooperate to provide semantic information sharing. The task of each server is to store and maintain ontologies and to provide data interchange facilities.

Service provider A uses his ontology server to develop a sports ontology (SO) (cf. 1). To make it a standard, A contacts a well-known ontology registry and registers SO there (cf. 2). He provides metadata about SO using an Ontology Meta-Ontology (OMO) (see Section 3). It is important to note that SO is stored in A's ontology

[1] http://kaon.semanticweb.org

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.

server. The registry doesn't store the ontology itself – it stores only metadata about the different available ontologies.

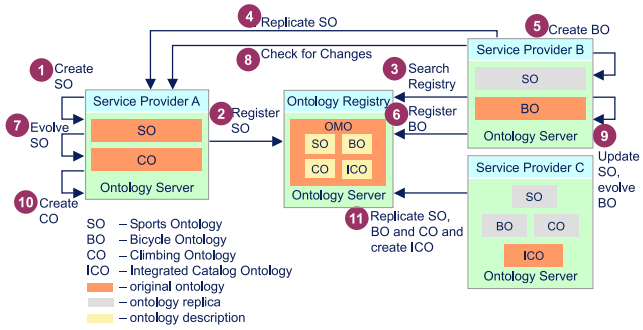

Figure 1: B2B Catalog Integration Scenario

Service provider B may now search the registry for ontologies that are appropriate to be used as a foundation for BO (cf. 3), performing a metadata-driven search for existing ontologies . To reuse SO, B replicates it from the A's ontology server into his ontology server (cf. 4). Replication is done to decouple subjects in the system. To develop BO (cf. 5), B uses the inclusion facilities of his ontology server, making all definitions from the SO available in BO (see Section 4). When done, B registers BO with the registry to enable reuse of BO (cf. 6).

As the business requirements change, A will change the SO[2] (cf. 7). At this stage the entire system is inconsistent – B hasn't yet updated BO or his local copy of SO. In the WWW context such situations simply cannot be avoided, due to numerous factors, such as the number, geographical distribution and technological heterogeneity of subjects involved, independence of their business cycles etc. Therefore, we must simply accept the fact that the service provider B works with an outdated version of the ontology – a request directed to B must be formulated using concepts from the version of SO that B currently understands, or the system will produce incorrect results. Although inconsistencies are inevitable, the infrastructure should provide means for their easier detection and management. In our case this means that B may check for newer ontology versions at leisure (cf. 8). When ready, he needs to determine which included ontologies have been updated and to apply the distributed ontology evolution algorithms (cf. 9) to bring both the BO and the local copy of SO up-to-date (see Section 5).

To demonstrate the complexity of the problem, let's assume that A also creates the climbing gear ontology (CO) based on SO (cf. 10). The service provider C combines all available ontologies into an integrated catalog ontology (ICO) based on both BO and CO (cf. 11). However, the dependencies now form a directed acyclic graph. This means that changes in the system must be propagated layer-by-layer. E.g., if SO changes, BO and CO must first be updated (in any order) and only then changes from both ontologies can be propagated to ICO.

3. ONTOLOGY REGISTRY

The scope of the Semantic Web spans various fields of human activities: business, research and cultural, to name just a few. Although seemingly isolated at first, these fields are often intertwined. Even in e-business, the same concept could be needed in completely different contexts (e.g. the BICYCLE concept can also be used to report the news on the Tour-de-France). To facilitate reuse between such radically different contexts, ontologies should be kept small and coherent, describing information that logically fits together. Processes and methodologies that promote such practices have been discussed in [20] and are not further elaborated in this paper. Making ontologies smaller and easier to reuse comes with a downside that the number of ontologies proliferates, so when a new ontology is needed, it is difficult to decide whether to build it from scratch, or whether an ontology can be reused. In order to maximize the reuse potential, a way of registering and locating well-known ontologies is needed.

[2]For example, A may refine the hierarchy by distinguishing sports utilities for professionals and for hobbyists

3.1 Describing Ontologies using OMO

An approach for locating ontologies has been discussed in [1] and is based on describing domain ontologies in a meta-ontology, containing information such as the ontology author, ontology location, used ontology language. This information is used by the ontology registry for providing functionalities such as searching and browsing.

We follow that approach and define the ontology meta-ontology (OMO), whose core structure is presented in Figure 2. It is an adaptation of the meta-ontology given in [1], extended with specific aspects required for management of multiple ontologies (see sections 4). Each registered ontology is represented as an instance of the OIMODEL concept, which contains information such as creation date, ontology name, etc. The creator of the ontology is represented either as an instance of PERSON or ORGANIZATION concepts. Further, an ontology may be associated with a particular project and an application field. The TERM concept is used to describe in more detail what the ontology is actually about – this is described in more detail in the next subsection.

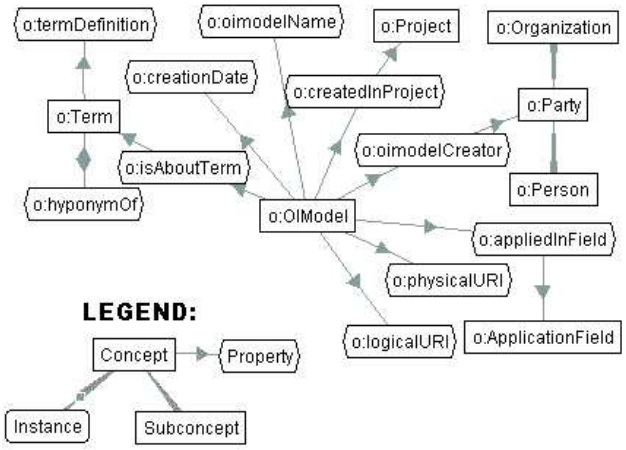

Figure 2: Ontology Meta-Ontology (OMO)

3.2 Improving Search Using WordNet

Locating ontologies described by meta-data certainly contributes to ontology reuse in the Web, but is not completely satisfactory. The main problem is that the content of the ontology isn't actually used for searching. Most users distinguish ontologies by their contents (e.g. about bicycles or climbing), which are often correlated with a couple of central concepts in the ontology. Hence, a natural criteria for the search is by the concepts contained in the target ontology.

This is actually more difficult to realize than it may seem at first. The fundamental problem is that the search condition should reference the ontological entities which are not known until the search

is executed. For example, the user may look for an ontology about bicycles. However, the concept BICYCLE is part of the ontology being sought for and will be known only when the search is executed. We call this the search bootstrapping problem.

Our solution to this problem is to match each ontology element against a well-known lexical semantic net, thus obtaining the ontology "digest" – a list of relevant ontology terms that is then included into the meta-ontology. The search for an ontology can than be expressed in terms of the lexical semantic net.

Currently, the most well-known lexical semantic net is WordNet[3] [7], whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, adjectives and adverbs are organized into so-called synonym sets (synsets), each representing one underlying lexical concept. Different types of relations link the synsets, e.g. the hyponym relation describing a specialization relationship between two synsets. We have converted WordNet[4] into our model [15] (which is discussed in more detail in section 4) as follows:

- Each synset is transformed into an instance of the SYNSET concept, which is in turn made a subconcept of the TERM concept.
- Synset words are transformed into instances of the SYNONYM concept and are attached to appropriate synset instances.
- Hyponym relationship between synsets are represented through a HYPONYMOF transitive property.

When a new ontology is registered at the registry, lexical information about concepts (labels, synonyms, etc.) is matched against synonyms in WordNet. Thus, for each concept a set of synsets – possible meanings – is obtained. In case too many possible meanings for a concept exist, the user is asked to disambiguate them manually.

It is important to understand that it is not the ontology itself that is stored into the registry – the registry will contain only meta-information about an ontology and references to chosen WordNet terms. In such a way a compact representation suitable for efficient search is obtained.

WordNet doesn't contain terms from all the fields of human activities. If the ontology is highly specialized, its terms may not exist in WordNet. For example, an ontology about mountain bicycles will likely contain the concept BOTTOM BRACKET, which doesn't exist in WordNet. Such terms will be generated automatically, added to WordNet and classified under the most general term. It is up to the registry maintainer to periodically reclassify these terms, manually or semi-automatically.

In [12] various methods for learning ontologies from texts are discussed and have been implemented in our infrastructure in form of the Text-To-Onto package. For example, if texts that describe the domain of registered ontologies are available, they can be used to extract possible suggestions for improvement of the hierarchy.

3.3 Searching the Registry

Searching the registry is performed by combining two different types of conditions. Query-by-example (QBE) is used for specifying conditions on the OMO definitions, supplying the constraints on various fields. Another part of the search condition consists of the keywords specifying the relevant terms that the ontology must contain. This has the following benefits:

- It is possible to find ontologies in a natural way by specifying terms that the ontology is about. This solves the search bootstrapping problem and increases the usability of the system.
- By using the hyponym links in the lexical semantic net, ontologies containing terms more specific than specified can be found.

The search is processed according to the following approach: Each search term is matched against the lexical semantic net and the list of possible meanings is retrieved. If the number of possible meanings exceeds a certain threshold, the user is presented with the list of possible meanings and asked to disambiguate the meanings. For each term a set of more specific terms is determined. Registered ontologies matching the search terms and subterms are retrieved. The average distance of the matched terms to the search terms is computed for each ontology. Ontologies are sorted in the increasing order of the average distance. Finally the list of ontologies is filtered against the supplied constraints on OMO fields.

For example, in our e-business scenario service provider B might issue a following search: the ORGANIZATIONNAME of the ontology is ANSI, the ontology is classified in the APPLICATIONFIELD of E-BUSINESS and the ontology references terms such as FRAME and BICYCLE. A possible result to this search will include ontologies containing the MOUNTAIN BIKE concept, since WordNet specifies the term MOUNTAIN BIKE as the hyponym BICYCLE.

4. ONTOLOGY REUSE

This section describes the issues enabling the reuse of ontologies located through the registry. In our approach we identified two basic building blocks for realizing reuse. First, *ontology inclusion* allows reusing ontologies available within the same node. Second, *ontology replication* enables inclusion in the case when ontologies are distributed on different ontology servers (nodes).

4.1 Ontology Inclusion

In traditional software systems significant attention is devoted to keeping modules well separated and coherent with respect to functionality, thus making sure that changes in the system are localized to a handful of modules. Reuse is seen as the key method in reaching that goal. One of the main focuses of all existing reuse mechanisms is completely eliminating the copy-and-paste reuse, which is seen as the prominent source of problems on software projects. Ontology-based systems in the Web are just a special class of software systems, so the same principles apply. If reuse is performed through duplication, problems arise when the reused ontology changes, as these changes must be applied on various multiple copies. Paraphrasing the open-closed reuse principle [14], each ontology should be a closed, consistent and a self-contained entity, but open to extensions in other ontologies.

These goals may be achieved by incorporating an explicit mechanism for including ontologies by reference into ontology languages and tools. Our approach is based on the ontology language given in [15]. Briefly, the ontology language is similar to existing Semantic Web standards, such as DAML+OIL or OWL, and is based on RDFS, but with clean separation of modelling primitives from the ontology itself (thus avoiding the pitfalls of self-describing primitives such as subClassOf) and incorporating several commonly used modeling primitives, such as transitive, symmetric and inverse properties. In order to preserve tractability and enable ontology evolution, in our approach we treat property domain and range specifications as constraints, not as axioms. We found this view to be intuitive and desired by most users having a strong background

[3]http://www.cogsci.princeton.edu/ wn/

[4]http://wim.fzi.de:8080/WordNet/WordNetAsInstance.zip

in object-oriented and database technologies. A distinguishing feature of our model is explicit support for modeling meta-classes and explicit modeling of lexical information. All information is organized in so-called OI-models (ontology-instance models), containing both ontology entities (concepts and properties) as well as their instances[5]. This allows grouping concepts with their well-known instances into self-contained units.

Reusing is supported by allowing an OI-model to include other OI-models, thus obtaining the union of the definitions from all included models. Cyclical inclusions are not allowed because evolution of cyclically dependent OI-model would be too difficult. Inclusion is performed by-reference – models are virtually merged, however, the information about the origin of each entity is represented explicitly. Currently we don't support resolving semantical heterogeneities between included models (e.g. establishing equivalences between the BICYCLE and the FAHRRAD concepts) – we plan to extend our approach to handle such cases in future.

Returning to the scenario from section 1, Figure 3 presents four example OI-models (SO – sports ontology, BO – bicycle ontology, CO – climbing ontology and ICO – integrated catalog ontology). BO and CO each include SO, thus gaining immediate access to all of its definitions. However, the information about the origin of ontology entities retained. Thus, the following distinctions exist:

- In SO and CO SPORTS UTILITY concept doesn't have any sub- or superconcepts. However, in BO it has one subconcept BICYCLE, and in ICO it has one subconcept and one superconcept CATALOG ITEM.
- Relationships between concepts also belong to appropriate OI-models. Hence, it is possible to establish that the subconcept relationship between the SPORTS UTILITY and the CATALOG ITEM is established in ICO.
- In SO the property USED IN has only SPORTS UTILITY as domain concept, whereas in ICO it has an additional domain concept POWER DRINK.

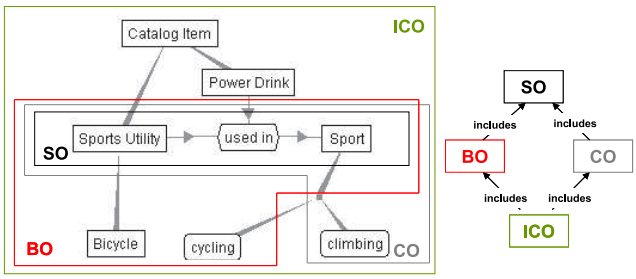

Figure 3: Ontology Reuse through Inclusion

On the right-hand side the direct acyclic inclusion graph between OI-models is shown. SO is indirectly included in ICO twice (once through BO and once through CO). However, ICO will contain all SO elements only once (in ICO there will be only one SPORTS UTILITY concept). The possibility of including an ontology through multiple paths has significant consequences on the ontology evolution, as discussed in subsection 5.2.

This example demonstrates the open-closed principle. Each OI-model keeps track about its own information and is a consistent, self-contained and closed unit. On the other hand, each OI-model is open to reuse, in which case any part of its structure can be extended, as long as the original model itself is not changed.

Our approach is currently limited to including entire models, rather than including subsets. Also, when a model is reused, information can only be added, and not retracted, and we currently don't deal with semantic inconsistencies between included ontologies. Although such advanced features may sometimes be useful, we deliberately limit our approach. By allowing inclusion of a part of a model it would be much more difficult to ensure the consistency of the including model, since it is not clear which additional elements from the original model must be included. For example, if USED IN property is not included in ICO, it is not clear how to treat instances having this property instantiated. Further, changing ontologies becomes more complex, because it is not clear how to propagate changes in ICO to SO.

4.2 Reusing Distributed Ontologies

Ontology inclusion allows reusing ontologies available within one node in the system. However, we envisage the Semantic Web where ontologies are spread across many different nodes, so the inclusion mechanisms cannot be used directly. There are two possible solutions how to achieve reuse in this case.

The first solution is to make all ontologies accessible through an ontology server, which could integrate the information from included ontologies virtually (on-the-fly) by accessing the servers of these ontologies. Such solution has the benefit that all changes in the included ontologies are immediately visible in the including ontologies. While this desirable feature increases the consistency, it has several serious drawbacks:

- Servers are tightly coupled – a failure of one system will cause failure of all servers that include the ontology.
- Standard top-level ontologies will be reused in many ontologies. Servers hosting them will therefore be overloaded, because they will be often contacted by many other servers.
- Because answering every query requires distributed processing, the performance of the system with today's infrastructure would be unacceptable.

Therefore, a more practical solution to the problem in the WWW context is to replicate distributed ontologies locally and to include them in other ontologies. Replication eliminates afore mentioned problems, but introduces significant evolution and consistency problems, which we further discuss in section 5. The most important constraint is that replicated ontologies should never be modified directly. Instead, the ontology should be modified at the source and changes should be propagated to replicas using the distributed evolution process.

The ontology replication should not be understood as simple copying of the original ontology. If this operation is performed in an ad-hoc way, evolving replicated ontologies will be impossible. Distributed evolution process requires associating meta-data with each ontology, which must be maintained during the replication process. This is described in further detail in in subsection 5.3. In order to replicate an ontology, it must be physically accessed. Ontologies on the Web are typically known under a well-known URI, which can be used to access the ontology through appropriate protocol (e.g. HTTP). However, this introduces problems when the ontology is replicated, since the URI used to access the ontology and the URI under which the ontology is originally known become different. To consistently handle this, we establish two different URIs for each ontology:

- The logical URI is unique for each ontology and is always the same, regardless of the ontology's location. The uniqueness

[5]In the rest of this paper we use the terms OI-model and ontology interchangeably.

of the URI is typically achieved by incorporating into it the Internet name of the organization that created the ontology.

- The physical URI unambiguously identifies the location of the ontology and contains all information necessary to access the ontology, such as the protocol to be used or relevant connection parameters.

For example, the SO from our example may have the logical URI http://www.sport.com/so. No other ontology with that URI exists anywhere in the world. However, the ontology may be replicated to the file system, and the physical URI will be file:/c:/so.kaon. If the ontology is stored in the database, then its physical URI may be jboss://wim.fzi.de:1099?http://www.sport.com/so.

To locate an ontology by the logical URI, this URI must be resolved to a physical URI through the ontology registry described in the previous section. After replication, the ontology registry is not needed any more, so the registry doesn't represent a single point of failure of the proposed system.

5. ONTOLOGY EVOLUTION

Ontology evolution can be defined as the timely adaptation of an ontology and consistent propagation of changes to the dependent artifacts. The complexity of ontology evolution increases as ontologies grow in size, so a structured ontology evolution process is required. Such a process has been described in [13]. The process starts with capturing changes either from explicit requirements or from the result of change discovery methods. Next, changes are represented formally and explicitly. The semantics of change phase prevents inconsistencies by computing additional changes that guarantee the transition of the ontology into a consistent state. In the change propagation phase all dependent artifacts (ontology instances on the Web, dependent ontologies and application programs using the changed ontology) are updated. During the change implementation phase required and induced changes are applied to the ontology in a transactional manner. In the change validation phase the user evaluates the results and restarts the cycle if necessary.

In this paper we extend this process towards handling multiple, distributed ontologies. As shown in Table 1, two dimensions of the overall ontology evolution problem may be identified.

		Nodes	
		One	Multiple
Ontologies	One	Single OE	-
	Multiple	Dependent OE	Distributed OE

Table 1: Levels of Ontology Evolution (OE) Problem

The first dimension defines the number of the ontologies being evolved, whereas the second specifies the physical location of evolved ontologies. Since it is not possible to fragment [17] one ontology across many nodes, we discuss ontology evolution at three levels. In subsection 5.1 we summarize the single ontology evolution problem. In subsection 5.2 we extend the change propagation and capturing phases to cover the evolution of multiple dependent ontologies within a single node. Finally, in subsection 5.3 we extend the change capturing and change implementation phases of the dependent evolution process to support evolution of distributed ontologies.

5.1 Single Ontology Evolution

For evolution of single ontologies the essential phase is the semantics of change phase, whose task is to maintain ontology consistency. Applying elementary ontology changes [19] alone will not always leave the ontology in a consistent state. For example, deleting a concept will cause subconcepts, some properties and instances to be inconsistent.

Definition 1. **[Single Ontology Consistency]** A single ontology is consistent if it satisfies a set of invariants defined in the ontology model [15] and if all used entities are defined.

Returning to the example in Figure 3, let's assume that the concept SPORTS UTILITY should be deleted. To prevent inconsistencies, before deleting it, the SPORTS UTILITY concept must be removed from the domain of the USED IN property. Since the ontology model definition specifies that properties without domain concepts aren't allowed, the property must be deleted as well. To do that, the SPORT concept must be removed from the range. The complete list of necessary changes obtained in the semantics of change phase is presented in the Figure 4:

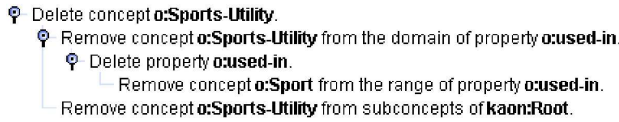

Figure 4: Generated Changes

For some change request, there may be multiple consistent resulting states. For example, when a concept from the middle of the concept hierarchy is deleted, all subconcepts may either be deleted or reconnected to other concepts. Further, the consistent state may be defined in multiple ways. For example, properties without domain and/or range concepts may or may not be considered inconsistent. Evolution strategies (see [19]) are used to customize the evolution process according to user's preferences.

5.2 Dependent Ontology Evolution

In this subsection we extend the single ontology evolution approach to take into account the inclusion relationships between ontologies within one node. An ontology that includes ontologies is called the dependent ontology. As the included ontology is changed, the consistency of the dependent ontology may be invalidated.

Definition 2. **[Dependent Ontology Consistency]** A dependent ontology is consistent if the ontology itself and all its included ontologies, observed alone and independently of the ontologies in which they are reused, are single ontology consistent.

Returning to the example of Figure 3, if the SPORTS UTILITY concept from SO is deleted, the ontology BO, and through transitivity of inclusion the ICO as well, will be inconsistent, since the BICYCLE and CATALOG ITEM concepts will have a parent concept and a child concept respectively, that are not defined. Moreover, it is important to notice that applying the deletion of the SPORTS UTILITY concept to the outer-most ontology (ICO) only is not sufficient. In ICO the USED IN property has two domain concepts, so removing one of them will not trigger the removal of the property. Therefore, if SO is considered independently, it is inconsistent, since the USED IN property will not have a domain concept in this ontology. The list of changes generated in this case through the single ontology evolution process is shown in Figure 5, which is quite different from the changes shown in Figure 4.

This example shows that maintaining consistency of a single ontology is not sufficient; dependent ontology consistency must be taken into account as well. This may be achieved by propagating

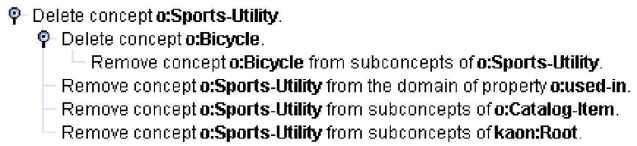

Figure 5: Generated Changes in ICO

changes from the changed ontology to all ontologies that include it. There are two ways of doing that [2]:

- *Push-based approach*: Changes from the changed ontology are propagated to dependent ontologies as they happen.
- *Pull-based approach*: Changes from the changed ontology are propagated to dependent ontologies only at their explicit request.

The pull-based approach is better suited for less stringent consistency requirements. Using this approach dependent ontologies may be temporarily inconsistent. This makes recovering of the consistency of dependent ontologies difficult, as the information about the original state of the changed ontology is lost. For example, when the concept SPORTS UTILITY is deleted, its position in the concept hierarchy is lost and is not available for resolution of inconsistencies of the BICYCLE concept in BO.

The push-based approach is suitable when strict dependent ontology consistency is required, since the information about the original state of the changed ontology is available for the evolution of the dependent ontology. For example, the removal of the concept SPORTS UTILITY requires previous resolution of the consistency of the BICYCLE concept in BO. We choose to take this approach, since in our target applications the permanent consistency of ontologies within one node is of paramount importance.

By adopting the push-based approach, there are three different strategies for choosing the moment when changes are propagated [18]. Using the periodic delivery, changes are propagated at regular intervals. Using ad-hoc delivery, changes are not propagated according to a previously defined plan. Both of these strategies are unacceptable for dependent ontology evolution, since they cause temporal inconsistencies of dependent ontologies. Therefore, we propagate changes immediately, as they occur.

We incorporate the push-based approach by extending the change propagation and change capturing phases of the single ontology evolution process as shown in Figure 6.

Figure 6: Dependent Ontology Evolution Process

The role of the Ontology Propagation Order component is to determine to which dependent ontologies the changes should be propagated, and in which order this should be done. The role of the Change Filtering component is to determine which changes must be propagated to which ontologies. The Change Ordering component determines the order in which changes must be received by each ontology.

Ontology Propagation Order. When propagating changes to dependent ontologies on a single node, the following three aspects relating to the ontology propagation order must be considered:

- As changes occur in an ontology, they must be pushed to all ontologies that either directly or indirectly (through other ontologies) include the changed ontology.
- In order to propagate a change to an ontology, the change must previously be processed by all ontologies on the path between the source and target ontology. Therefore, all ontologies on a single node are topologically sorted[6] according to their inclusion relationship. The topological order organizes the dependent ontologies in such a way that for each O1 and O2, if O1 includes O2 directly or indirectly, then O2 occurs before O1 in a linear ordering.
- Since all ontologies at a node are topologically ordered, when changes are propagated to dependent ontologies, only those ontologies that include the changed ontology and that follow the changed ontology in the topological order must be visited. Note that if cyclical inclusions of OI-models were allowed, the propagation order would contain cycles and would be extremely hard to manage.

Returning to the example in Figure 3, changes in SO must be propagated to BO, CO and ICO (since ICO includes SO indirectly though BO and CO). Further, several different topological orders may exist (e.g. SO, BO, CO, ICO or SO, CO, BO, ICO), since some ontologies are independent on each other (BO and CO). The propagation of changes must be performed in either one of these orders. On the other hand, assuming the first topological order (SO, BO, CO and ICO) a change in BO is propagated only to ICO – although CO is after BO in the topological sort, it doesn't include BO so it doesn't receive BO's changes.

Change Filtering. As a change from the source ontology S is propagated to a dependent ontology D, in order to maintain the consistency of D, additional changes will be generated as explained in subsection 5.1. These changes must also be propagated further up the ontology inclusion topological order. However, only induced changes should be forwarded. If original changes were propagated as well, then ontologies that include D would receive the same change multiple times: directly from S and indirectly from all ontologies on any inclusion path between D and S. This would result in an invalid ontology evolution process, since the same change cannot be processed twice. In order to prevent that, propagated changes are filtered.

As shown in Figure 5, deletion of the SPORTS UTILITY concept in SO is propagated to BO resulting in new changes: the removal of the BICYCLE concept as the subconcept of the SPORTS UTILITY concept and the removal of the BICYCLE concept itself (if the evolution strategy [19] requires the removal of the orphaned concepts). Only these two changes are propagated to ICO. Removal of the concept SPORTS UTILITY is propagated to ICO from SO directly and must not be propagated from BO. Notice that change filtering is not done for the sake of performance: if SPORTS UTILITY were propagated to ICO from BO as well, then ICO would receive the same change twice, and the second change would fail, since the concept has already been deleted.

Change Ordering. The order of processing changes in each ontology is important. Let's assume that S is the ontology being changed, I is some ontology that directly includes S and D is some ontology that directly includes I. It is important that D processes changes generated by I before changes generated by S. Otherwise, if D receives changes from S before changes from I, S's changes will generate additional changes in D that include those that will later be received from I. This in turn will also lead to processing the same change twice. This approach is recursively applied when D and S are connected with paths of length greater than two.

[6]The topological order of a directed graph is an ordering of graph's nodes where each node occurs after all of its predecessors.

Returning to the example in Figure 3, ICO should process the removal of the SPORTS UTILITY concept after processing the removal of the subconcept BICYCLE from BO. If this were not the case, processing removal of SPORTS UTILITY in ICO would generate removal of the subconcept BICYCLE in ICO, which will then be later received from BO.

The algorithm for evolution between dependent ontologies within one node is presented in Algorithm 1. It processes all changes that are requested by the user through the procedure PROCESSCHANGE (cf. 2–4). This procedure resolves a change by generating the additional changes needed to keep the consistency of the ontology *o* for which the method was called (cf. 7–9). Only changes generated in *o* are propagated (cf. 11) to the all ontologies including *o* according to the topological order of all ontologies within the node (cf. 13–14). The recursive call (cf. 16) of the PROCESSCHANGE procedure for the filtered change and topological order of dependent ontologies guarantees that the receiving ontologies will process the changes from the directly included ontologies before changes from the indirectly included ontologies. Finally, the change is applied to the ontology *o* (cf. 21).

Algorithm 1 Dependent Ontology Evolution Algorithm

EVOLVEONTOLOGIES($\mathcal{LC}$, o)
Require: $\mathcal{LC}$ - list of changes, o - ontology being changed
1: $\mathcal{TS}$ = topological sort of ontologies at the node
2: **for all** $c \in \mathcal{LC}$ **do**
3: PROCESSCHANGE(c, o)
4: **end for**

PROCESSCHANGE(c, o)
Require: c - change to process, o - ontology being changed
5: es = evolution strategy for o
6: */*Semantics of Change*/*
7: **while** generated change gc by es for c in o **do**
8: processChange(gc, o)
9: **end while**
10: */*Change Filtering*/*
11: **if** c is generated in o **then**
12: */*Ontology Propagation Order*/*
13: **for all** ontology d after o in $\mathcal{TS}$ **do**
14: **if** ontology d includes o **then**
15: */*Change Ordering*/*
16: processChange(c, d)
17: **end if**
18: **end for**
19: **end if**
20: */*Change Implementation*/*
21: change ontology o according to c

5.3 Distributed Ontology Evolution

A distributed dependent ontology is an ontology that depends on an ontology residing at a different node on the network. The physical distribution of ontologies is very important, since it creates additional problems that are not encountered when the ontologies are collocated. This additional complexity stems from the fact that reusing distributed ontologies is achieved through replication (see section 4). Since the original ontology is updated autonomously and independently of replicas, this in turn introduces an additional type of ontology consistency.

Definition 3. **[Replication Ontology Consistency]** An ontology is replication consistent if it is equivalent to its original and all its included ontologies (directly and indirectly) are replication consistent.

To explain this notion we assume a distributed system of replicated ontologies as shown in Figure 1. Ontology SO at service provider B is replication inconsistent if it hasn't been updated according to changes in its original at the service provider A. This implies the replication inconsistency of BO at provider B (since BO includes SO which is replication inconsistent). Finally, this implies the replication inconsistency of ICO at the service provider C in the same way.

To resolve replication inconsistencies between ontologies, first a way of synchronizing distributed ontologies is needed. Table 2 discusses the pros and cons of two well-known approaches [2] for synchronizing distributed systems. Although seemingly similar, there is significant difference to the approaches described in subsection 5.2, as here we address the fact that we are dealing with a distributed system.

	Push	Pull
Dependency Information	centralized	local
Complexity of management	high	medium
Type of consistency	strict	loose
Communication overhead	high	optimized

Table 2: Push vs. Pull Synchronization of Ontologies

Under push synchronization the changes of originals are propagated to ontologies including replicas immediately. We identify several drawbacks of using this approach for realistic scenarios on the Web. First, to propagate changes, for each ontology information about ontologies that reuse it should be available. Thus, an additional centralized component managing inclusion dependencies between ontologies is needed. Second, with the increase in the number of ontologies and of subjects reusing them, the number of dependencies will grow dramatically. Managing them centrally will be too expensive and impractical as the problem of evolving dependencies is raised. Third, forcing all ontologies to be "strictly" consistent at all times reduces the possibility to express diversities in a huge space such as the Web. Subjects on the Web may not be ready to update their dependent ontologies immediately and may opt to keep the older version deliberately. Finally, the changes are propagated one-by-one, thus introducing significant communication overhead. Grouping changes and sending them on demand will perform better.

Therefore, in the distributed environment we advocate using the pull synchronization. Under this approach information about included ontologies is stored in the dependent ontology, thus eliminating the need for central dependency management. Original ontologies are checked periodically to detect changes and collect deltas. During this process, it may be possible to analyze changes and to reject them if they don't match the current needs. Thus, we propose a "loosely" consistent system, since replication consistencies are enforced at request. Permitting temporary inconsistencies is a common method of increasing performance in distributed systems [17]. Hence, we use the pull approach for synchronizing originals and replicas, whereas we use the push approach for maintaining consistency of ontologies within one node. Thus, our solution employs a hybrid synchronization strategy combining their favorable features while avoiding their disadvantages.

Regardless of the synchronization approach, a question about how to resolve replication inconsistencies remains open. We note that replication inconsistencies cannot be resolved by simply replacing the replica with the new version of the original. This will

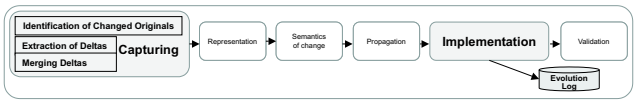

Figure 7: Distributed Ontology Evolution Process

cause inconsistencies of the dependent ontologies, as discussed in subsection 5.2. Instead, replication and dependency inconsistency must be resolved together in one step. This can be achieved by applying dependent evolution algorithms on deltas – changes that have been applied to the original since the last synchronization of the replica. By using the pull synchronization strategy and by applying the dependent evolution process from Figure 6 to deltas, we derive the distributed ontology evolution process through three extensions. This process, shown in Figure 7, is responsible for propagating changes from originals to replicas. We extend the implementation phase by introducing the evolution log for keeping information about performed changes. Further, we extend the change capturing phase by three components. During identification of changed originals we identify which original ontologies have changed. In extraction of deltas we identify the changes performed at the original and not at the replica by reading the evolution log. Finally, during merging of deltas we generate a cumulative list of changes that must be performed at the replica.

5.3.1 Logging Changes

In order to resolve replication inconsistencies, two known ways of identifying deltas between originals and replicas are known [17]: (1) the full content of the original ontology may be compared to the replica; (2) the history of changes to the original may be kept explicit. The first solutions requires extracting changes from differences between the original and the replica which is a complicated and time-consuming process. Further, to compare ontologies the current version of the original must be copied temporarily to the replica's node. This may incur unnecessary communication overhead. For example, the WordNet ontology described in section 3 is very large. If a concept is added to it, it is better to transfer only the information about this addition, instead of transferring the whole ontology.

To avoid these drawbacks, we follow for the second option. For each distributed ontology an instance of a special evolution log ontology is created, which tracks the history of changes to the ontology. Apart from the distributed evolution, the evolution log is also used to provide the following additional capabilities:

- Users may often want to undo the changes to the ontology. For each elementary change a sequence of inverse changes may be derived that completely undo the original changes. Hence, by applying inverse changes in reverse order any previous state of an ontology may be reconstructed [19].
- With each change additional meta-information may be associated. This information can serve as a source for different knowledge discovery methods, e.g. mining about change trends.

The evolution log ontology, shown in Figure 8, models what changes, why, when, by whom and how are performed in an ontology. Each change is represented as an instance of one of the subconcepts of the CHANGE concept. The structure of the hierarchy of change types reflects the underlying ontology model by including all possible types of changes (e.g. ADDENTITY, REMOVEENTITY, etc.). Additional information, such as the date and time of change, as well as the identity of the change initiator may be associated through appropriate properties. Information supporting decision-making, such as cost, priority, textual description of the reason for change etc. may also be included. Entities from the ontology being changed are related to instances of the CHANGE concept through HAS_REFERENCEENTITY property. As described previously, elementary changes may cause new changes to be introduced by the evolution strategy in order to keep the ontology consistent – such dependencies may be represented using the CAUSECHANGE property. Groups of changes of one request are maintained in a linked list using the HAS_PREVIOUSCHANGE property.

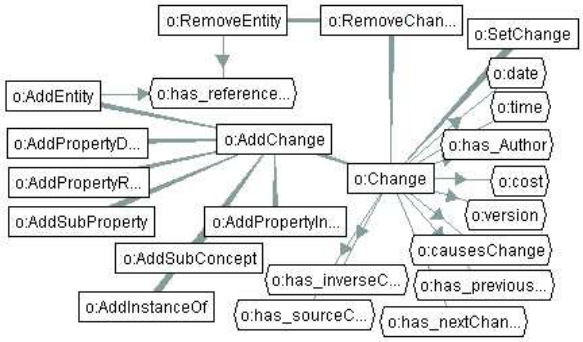

Figure 8: A part of the evolution ontology

5.3.2 Resolving Replication Inconsistencies

As shown in Figure 7, resolving replication inconsistencies is performed through three additional components. Resolving replication inconsistencies is initiated by specifying an original whose included replicas should be updated and is performed as follows.

Identification of Changed Originals. This step first checks whether resolution of replication inconsistency can be performed at all. If for some directly included replica the original has replication inconsistency, then the process is aborted. Otherwise, a list of directly included replicas having pending replication inconsistency (but whose original is replication consistent) is determined. Since the dependent ontology consistency for the ontologies on the same node is required, this approach is recursively applied on the all ontologies that include the ontology whose replication inconsistency is resolved.

Let's assume that the service provider C from Figure 1 wants to resolve the replication inconsistency of ICO. Its directly included replicas, namely BO and CO are examined. For each of them the replication consistency of the original is checked. If BO at the service provider B has replication inconsistency (due to changes from A in SO which haven't been applied at B's replica of SO), then the process is aborted. If BO at the service provider B is replication consistent, but BO at service provider C is not (since BO at B has been changed), then BO is scheduled for further analysis. The consistency of the BO's original is required since ICO will obtain changes from SO through BO's and CO's evolution log.

In order to optimize this step, the set of directly included ontologies to be taken into account may be reduced by eliminating all directly included ontologies that are available through some other paths. In the case that the ontology ICO directly includes the ontology SO, the ontology SO would be eliminated from the further consideration, since it can be obtained through BO and CO.

Replication consistency is performed by determining the equivalence of the ontology with its original and by recursively determining the replication consistency of included ontologies. The following information is needed to perform that:

- Each ontology contains a physical URI of its original.
- Each ontology contains a physical URI of its evolution log.
- Each ontology has a version number associated with it that is incremented each time when an ontology is changed. Thus checking the equivalence of the replica and the original can be done by simple comparison of that number.

Extraction of Deltas. After determining directly included replicas to be updated, the evolution log for these ontologies is accessed. The location of the evolution log is specified within each ontology and is copied to replicas. For each log the extracted deltas contain all changes that have been applied to the original after the last update of the replica, as determined by the version numbers.

Merging Deltas. Deltas extracted from evolution logs in the previous step are merged into a unified list of changes. Since an ontology can be included in many other ontologies, its changes will be included into evolution logs of all of these ontologies. Hence, the merge process must eliminate duplicates. Also, changes from different deltas caused by the same change from a common included ontology should be grouped together.

For example, if the ontology SO is changed, the evolution logs of the BO and CO will contain these changes, as well as their own extensions. Hence, when changes from BO's and CO's logs are merged in order to update ICO, the changes to SO will be mentioned twice. Thus, only one change to SO should be kept while discarding all others. However, changes in BO and CO caused by a change in SO must be grouped together.

These three steps result in an integrated list of changes that must be processed at the target node using the dependent ontology evolution process as discussed in subsection 5.2.

6. IMPLEMENTATION

The presented infrastructure is implemented within KAON - an ontology management infrastructure developed at FZI and AIFB at the University of Karlsruhe. We consider reusing existing ontologies as an integral part of ontology engineering. Thus, interaction with the registry should be an integral part of the engineering environment. Because of that, we decided against a more common approach of providing a Web-based interface to the registry. Instead, we integrated the registry with OI-modeler – a stand-alone application for modular ontology engineering. To support usage of our application in the Web we provided a Java WebStart interface for OI-modeler[7]. A snapshot of registry usage is given in Figure 9. On the left-hand side there is OI-modeler with the SO being edited in it. On the right-hand side registration information about SO is presented.

The central component of KAON is the KAON API providing programmatic access to ontologies and is described in [15]. Briefly, the API decouples the applications from the persistent storage. It provides API implementations for accessing RDF(S) files and for storing OI-models in relational databases. The inclusion facilities described in section 4 have been implemented within the API. Ontology evolution is also implemented within the API and is described in more detail in [19]. This implementation has been extended for dependent and distributed ontology evolution.

7. RELATED WORK

An excellent comparison of existing ontology libraries is given in [4]. Many of these systems (WebOnto, Ontolingua, SHOE and

[7]http://wim.fzi.de:8080/kaon/kaon_workbench.jnlp

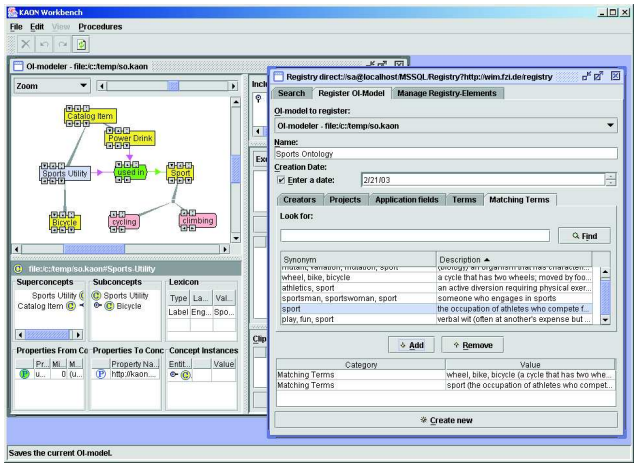

Figure 9: Ontology Management and Registry Screenshot

Ontology Server) combine the storage, searching and management facilities and provide means for editing ontologies and reasoning with them. Our infrastructure is different in that our approach envisages many servers containing only several ontologies. On top of that we provide centralized registries for searching ontologies and distributed evolution algorithms for ontology management.

Our ontology registry may be compared to the DAML Ontology Library, $(Onto)^2 Agent$ [1] and DCMI Registry [9]. These systems don't provide means for ontology storage, but only contain pointers to actual ontologies. DAML Ontology Library and DCMI Registry provide the means for searching ontologies based on the ontology contents. However, the search is based on lexical matching of element names. Our approach establishes links between concepts in different ontologies by matching each ontology against WordNet, thus providing more intuitive searching.

Reusing ontologies in the Semantic Web is hindered by the fact that RDF(S) doesn't provide means for including elements from other ontologies. Within RDF(S) there is no notion of the model representing a subset of the statement on the Web. Instead, each RDF fragment can freely refer to any resource defined anywhere on the Web. This presents serious problems to tool implementors, since it is not possible to reason over the entire Web. Recognizing this shortcomings, many ontology languages, e.g. OIL [8], DAML+OIL [3] and OWL [5], to mention just a few of them, provide means for declarative inclusion of other models.

However, most tools simply use these declarations for reading several files at the beginning and then create an integrated model. OilEd – a tool for editing OIL and DAML+OIL ontologies developed at the University of Manchester – does exactly that: importing an ontology actually inserts a copy of the original ontology into the current ontology. As mentioned before in this paper, this has drawbacks related to ontology evolution. On the other hand, tools such as Ontolingua [6], offer support even for cyclical ontology inclusion. However, to the best of authors' knowledge, these tools don't provide evolution of included ontologies. Protege-2000 [16] – a widely used tool for ontology editing developed at Stanford – provides the best support for ontology inclusion so far. In Protege it is possible to reuse definitions from a project by including an entire project. However, the implemented inclusion mechanism is too crude, as it doesn't allow extension of included entities. For example, it is not possible to re-classify or add a slot to a class in the including model. Further, only the outermost model may be changed, thus making the evolution of dependent ontologies impossible.

Although evolution over time is an essential requirement for successful application of ontologies, methods and tools to support this complex task completely are missing. There are very few approaches investigating that problem. In [10] it is pointed out that ontologies on the Web will need to evolve and a new formal definition of ontologies for the use in dynamic, distributed environments is provided. While multiple version of ontologies and ontology reuse is supported, the change propagation between distributed and dependent ontologies is not treated. In [11] the authors describe a system that provides support for the versioning of ontologies. In contrast to that approach, which detects changes by comparing ontologies, we track information about all performed changes, since the change detection is a complicated and a time-consuming process. Further, it is impossible to determine the cause and the consequences of a change, which is a crucial requirement for the consistency of the dependent ontologies. Moreover, research in distributed ontology evolution can also benefit from the research in distributed systems [17]. In [2] the authors describe techniques that combine push and pull synchronization in an intelligent and adaptive manner while offering good resiliency and scalability. We extend this approach by taking into account not only the coherency maintenance of the cached data but the maintenance of the dependent and replication consistency as well.

8. CONCLUSION

In this paper we have discussed an infrastructure for searching, reusing and evolving ontologies in a distributed environment. We see registering and searching existing ontologies as key to enabling wide-spread ontology reuse. Our approach focuses on ontology inclusion going beyond the simple cut-and-paste inclusion. Distributed ontology reuse is supported through controlled ontology replication, which is necessary under present technological constraints, such as available bandwidth. Evolving ontologies that reuse other ontologies is a complex problem. We first considered evolving dependent ontologies within one node. Further, we extended this solution to the distributed case. We advocate the pull synchronization mechanism, in order to keep the autonomy of each node in the system. Finally, we presented an approach for evolution of distributed ontologies, which is based on keeping change information available in form of evolution logs. The overall approach has been implemented within KAON.

Our future work will be directed towards relaxing the constraint that only entire ontologies may be reused. Lifting these constraints will have significant impact on the evolution of dependent ontologies. Furthermore, we will focus on various problems of integrating semantically heterogeneous ontologies.

9. REFERENCES

[1] J. Arpirez, A. Gomez-Perez, A. Lozano, and H. S. Pinto. $(ONTO)^2 Agent$: An ontology-based WWW broker to select ontologies. In *Proc. 13th European Conf. on Artificial Intelligence – ECAI'98*, Brighton, England, August 1998.

[2] M. Bhide, P. Deoasee, A. Katkar, A. Panchbudhe, and K. Ramamritham. Adaptive Push-Pull: Disseminating Dynamic Web Data. *IEEE Transaction on Computers*, June 2002.

[3] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. DAML+OIL (March 2001) Reference Description, http://www.w3.org/TR/daml+oil-reference.

[4] Y. Ding and D. Fensel. Ontology Library Systems: The Key to Successful Ontology Re-Use. In *Proc. 1st Int'l Semantic Web Working Symposium (SWWS'01)*, 2001.

[5] M. Dean et al. OWL Web Ontology Language 1.0 Reference, W3C Working Draft 29 July 2002, http://www.w3.org/TR/owl-ref/.

[6] A. Farquhar, R. Fikes, and J. Rice. The Ontolingua server: Tools for collaborative ontology construction. Technical report, Stanford KSL 96-26, September 1996.

[7] C. Fellbaum. *WordNet – An electronic lexical database*. MIT Press, 1998.

[8] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a Nutshell. In *Proc. 12th Int'l Conf. on Knowledge Engineering and Knowledge Management (EKAW-2000)*, Juan-les-Pins, France, October 2000.

[9] R. Heery and H. Wagner. A Metadata Registry for the Semantic Web. *D-Lib Magazine*, 8(5), 2002.

[10] J. Heflin and J.A. Hendler. Dynamic Ontologies on the Web. In *Proc. 7th Nat'l Conf. on Artificial Intelligence AAAI-2000*. AAAI/MIT Press, 2000.

[11] M. Klein, A. Kiryakov, D. Ognyanov, and D Fensel. Ontology Versioning and Change Detection on the Web. Siguenza, Spain, October 2002.

[12] A. Maedche. *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers, 2002.

[13] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. Ontologies for Enterprise Knowledge Management. *To appear in: IEEE Intelligent Systems*, 2003.

[14] B. Meyer. *Object-oriented Software Construction (2nd Edition)*. Prentice Hall, 1997.

[15] B. Motik, A. Maedche, and R. Volz. A Conceptual Modeling Approach for Semantics-driven Enterprise Applications. In *Proc. 1st Int'l Conf. on Ontologies, Databases and Application of Semantics (ODBASE-2002)*, October 2002.

[16] N. F. Noy, R. W. Fergerson, and M. A. Musen. The knowledge model of Protege-2000: Combining interoperability and flexibility. In *Proc. 12th Int'l Conf. on Knowledge Engineering and Knowledge Management (EKAW-2000)*, Juan-les-Pins, France, October 2000.

[17] M.T. Oezsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall International, Inc., 1999.

[18] G. Pierre and M. van Steen. Dynamicaly Selecting Optimal Distribution Strategies on Web Documents. *IEEE Transaction on Computers*, 2002.

[19] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven Ontology Evolution Management. In *Proc. 13th European Conf. on Knowledge Engineering and Knowledge Management (EKAW-2002)*, Siguenza, Spain, October 2002.

[20] Y. Sure, J. Angele, and S. Staab. Guiding Ontology Development by Methodology and Inferencing. In *Proc. 1st Int'l Conf. on Ontologies, Databases and Application of Semantics (ODBASE-2002)*, October 2002.