Finding and characterizing changes in ontologies

Michel Klein[1], Atanas Kiryakov[2], Damyan Ognyanov[2], and Dieter Fensel[1]

[1] Vrije Universiteit Amsterdam
{michel.klein|dieter}@cs.vu.nl
[2] OntoText Lab. Sofia
{naso|damyan}@sirma.bg

Abstract. Recently, the interest in the use of ontologies — which can be seen as formal representations of conceptual models — has increased because of the excitement about the vision of a "Semantic Web". When ontologies are used on the web, the distributed and dynamic nature of it requires advanced support for change management. This paper discusses the working of OntoView, a web-based change management system for ontologies. OntoView provides a transparent interface to different versions of ontologies, by maintaining not only the transformations between them, but also the conceptual relation between concepts in different versions. It uses several rules to find changes in ontologies and it visualizes them — and some of their possible consequences — in the file representations. The user is able to specify the conceptual implication of the differences, which allows the interoperability of data that is described by the ontologies. This paper briefly describes the system and presents the mechanism that we used to find and classify changes in RDFS / DAML ontologies. It also shows how users can specify the conceptual implication of changes to help interoperability.

1 Ontologies as conceptual models on the Web

Ontologies are specifications of a formal and common understanding of a domain, which try to reduce the gap between the way in which humans and machines handle information. They are developed for knowledge sharing and reuse (see [8]). In the last few years, there has been a lot of interest in ontologies. The current excitement about the vision of a Semantic Web [4] form an additional stimulant for the interest in ontologies. In this vision, ontologies have a role in defining and relating concepts that are used to describe data on the web. The Semantic Web is the idea of having data on the Web defined and linked in a way that it can be used for more effective discovery, automation, integration, and reuse across various applications[3].

The distributed and dynamic character of the web emphasizes a specific issue of ontology research: the evolution and versioning of ontologies. Ontologies are often developed by several persons and continue to evolve over time. Moreover, domain changes, adaptations to different tasks, or changes in the conceptualization might cause modifications of the ontology. This will likely cause incompatibilities in the applications and ontologies that refer to them, and will give wrong interpretations to data or make data inaccessible [9].

[3] http://www.w3.org/2001/sw/Activity

To handle ontology changes, a change management system is needed that keeps track of changes and versions of ontologies. Moreover, it is necessary to maintain the links between the versions and variants that specify the relations and updates between the versions. These links can be used to re-interpret data and knowledge under different versions. The ontologies and their relations together form a *web* of ontologies. The specification of these links is thus very important.

In this paper, we present a web-based system that supports the user in specifying the conceptual relation between version of concepts. The system, called OntoView, also maintains those links, together with the transformations between them. It use them to provide a transparent interface to different versions of ontologies, both at a specification level as at a conceptual level. It can also export the differences between versions as separate "mapping ontologies", which can be used as adapters for the re-interpretation of data and other ontologies.

The system could be used in a scenario where ontologies are used to describe data on the internet. Users can copy and adapt ontologies of others, e.g. to fulfil their specific needs. After a specific user changed the ontology off-line, he can use the system to compare the new version of the ontology with the old one and characterize the conceptual implications of changes. The system then exports the conceptual relations between the concepts in the different versions together with meta-data. This exported mapping ontology can be used to translate or to reinterpret instance data or other ontologies automatically.

Most of the ideas underlying the system are not depending on a specific ontology language. However, the implementation of specific parts of the system will be dependent on the used ontology language, for example the mechanism to detect changes. Throughout this article, we will use DAML+OIL[4] [6,7] and RDF Schema (RDFS) [5] as ontology languages. These two languages are widely considered as basis for future ontology languages for the Web.

The rest of the paper is organized as follows. In the next section, we discuss some issues about update relations between ontologies. In section 3, we give a brief overview of the versioning system and describe its the main functions. Section 4 describes the main feature of the system: comparing ontologies. In that section, we explain the mechanism we used to find changes in RDF-based ontologies and present some of the rules that we used to encode change types. Finally, we conclude the paper in section 5.

2 The update relation between ontologies

There are two important things to discuss when considering an update relation. First, this is the difference between update relations and conceptual relations inside an ontology.

Ontologies usually consist of a set of class (or concept) definitions, property definitions and axioms about them. The classes, properties and axioms are related to each other and together form a model of a part of the world. We call these relations conceptual relations inside the domain of interest. A change constitutes a new version of the

[4] Available from `http://www.daml.org/language/`

ontology and defines an orthogonal update relation between the original version of the ontology and the new version. This update relation between two ontologies also entails update relations between concepts and properties in the old version and those in the new version. These implied update relations are depicted in Figure 1.

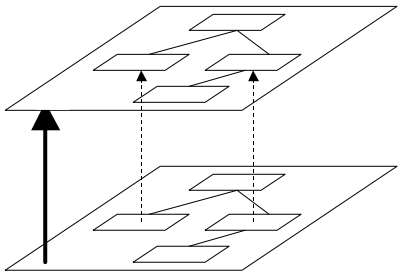

Fig. 1. An update relation (thick arrow) and the implied conceptual relations (dashed arrows) between classes in two version of an ontology.

The update relations between two versions of a concept, e.g. between class $A_{1.0}$ and class $A_{2.0}$, are more than pure conceptual relation in the domain of interest. The update relation also describes meta-information about the change of the concept. We can distinguish the following properties of an update relation:

- **transformation** or **actual change**: a specification of what has actually changed in an ontological definition, specified by a set of change operations (cf. [1]), e.g., change of a restriction on a property, addition of a class, removal of a property, etc.;
- **conceptual relation**: the logical relation between constructs in the two versions of the ontology, e.g., specified by equivalence relations, subsumption relations, or logical rules;
- descriptive meta-data like **date**, **author**, and **intention** of the update: this describes the when, who and why of the change;
- **valid context**: a description of the context in which the update is valid. In its simplest form, this might consist of the date when the change is valid in the real world, conform to *valid date* in temporal databases [12] (in this terminology, the "date" in the descriptive meta-data is called *transaction date*). More extensive descriptions of the context, in various degrees of formality, are also possible.

A well-designed ontology change specification mechanism should take all these characteristics into account.

Another issue to discuss about ontology updates is the possible discrepancy between changes in the specification and changes the conceptualization. We have seen that a ontology is a *specification* of a *conceptualization*. The actual specification of concepts and properties is thus a *specific representation* of the conceptualization: the same concepts could also have been specified differently. Hence, a change in the specification does

not necessarily coincide with a change in the conceptualization [9], and changes in the specification of an ontology are not per definition ontological changes.

For example, there are changes in the definition of a concept which are not meant to change the concept itself: attaching a slot "fuel-type" to a class "Car". Both class-definitions still refer to the same ontological concept, but in the second version it is described more extensively. Theoretically, the other way around is also possible: a concept could change without a change in its specification. However, this usually means that the concept is badly modelled.

It is important to distinguish changes in ontologies that affect the conceptualization from changes that don't. In [13] the following terms are used:

- **conceptual change**: a change in the way a domain is interpreted (conceptualized), which results in different ontological concepts or different relations between those concepts;
- **explication change**: a change in the way the conceptualization is specified, without changing the conceptualization itself.

A specific modification of an ontology cannot automatically be classified as belonging to one of these categories, because it is basically a decision of the modeler. However, heuristics can be applied to suggest the effects of changes. We will discuss that later on.

3 General description of OntoView

OntoView is a web-based system under development that provides support for the versioning of online ontologies, which might help to solve some of the problems of evolving ontologies on the web. Its main function is to help the a user to manage changes in ontologies and keep ontology versions as much interoperable as possible. It does that by comparing versions of ontologies and highlighting the differences. It then allows the users to specify the conceptual relation between the different versions of concepts. This function is described more extensively in the next section.

The system can also function as a storage system for version of ontologies, providing a transparent interface to arbitrary versions of ontologies. To achieve this, the system maintains an internal specification of the relation between the different variants of ontologies, with the aspects that were defined in section 2: it keeps track of the **meta-data**, the **conceptual relations** between constructs in the ontologies and the **transformations** between them.

OntoView is inspired by the Concurrent Versioning System CVS, which is used in software development to allow collaborative development of source code. The first implementation is also based on CVS and its web-interface CVSWeb[5]. However, during the ongoing development of the system, we are gradually shifting to a complete new implementation that will be build on a solid storage system for ontologies, e.g., Sesame[6].

Besides the ontology comparison feature, the system has the following functions:

[5] Available from `http://stud.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/`
[6] A demo is available at `http://sesame.aidministrator.nl`

– **Reading changes and ontologies.** OntoView will accept changes and ontologies via several methods. Currently, ontologies can be read in as a whole, either by providing a URL or by uploading them to the system. The user has to specify whether the provided ontology is new or that it should be considered as an update to an already known ontology. In the future, OntoView will also accept changes by reading in transformations, mapping ontologies, and updates to individual definitions. These update methods provides the system with different information than the method described above. For that reason, this also requires an adaptation of the process in which the user gives additional information.
– **Identification.** Identification of versions of ontologies is very important. Ontologies describe a consensual view on a part of the world and function as reference for that specific conceptualization. Therefore, they should have a unique and stable identification. A human, agent or system that conforms to a specific ontology, should be able to refer to it unambiguously.
– **Analyzing effects of changes.** Changes in ontologies do not only affect the data and applications that use them, but they can also have unintended, unexpected and unforeseeable consequences in the ontology itself [11].

  OntoView provides some basic support for the analysis of these effects. First, on request it can also highlight the places in the ontology where conceptually changed concepts or properties are used. For example, if a property “hasChild” is changed, it will highlight the definition of the class “Mother”, which uses the property “hasChild”. In the future, this function should also exploit the transitivity of properties to show the propagation of possible changes through the ontology.

  Further, we expect to extend the system with a reasoner to automatically verify the changes and the specified conceptual relations between versions. For example, we could couple the system with FaCT [3] and exploit the Description Logic semantics of DAML+OIL to check the consistency of the ontology and look for unexpected implied relations.
– **Exporting changes.** The main advantage of storing the conceptual relations between versions of concepts and properties is the ability to use these relations for the re-interpretation of data and other ontologies that use the changed ontology. To facilitate this, OntoView can export differences between ontologies as separate mapping ontologies, which can be used as adapters for data sources or other ontologies. They only provide a partial mapping, because not all changes can be specified conceptually, e.g. complicated changes like splits of concepts, or deletions.

4 Comparing ontologies

One of the central features of OntoView is the ability to compare ontologies at a structural level. The comparison function is inspired by UNIX `diff`, but the implementation is quite different. Standard `diff` compares file version at line-level, highlighting the lines that textually differ in two versions. OntoView, in contrast, compares version of ontologies at a *structural* level, showing which definitions of ontological con-

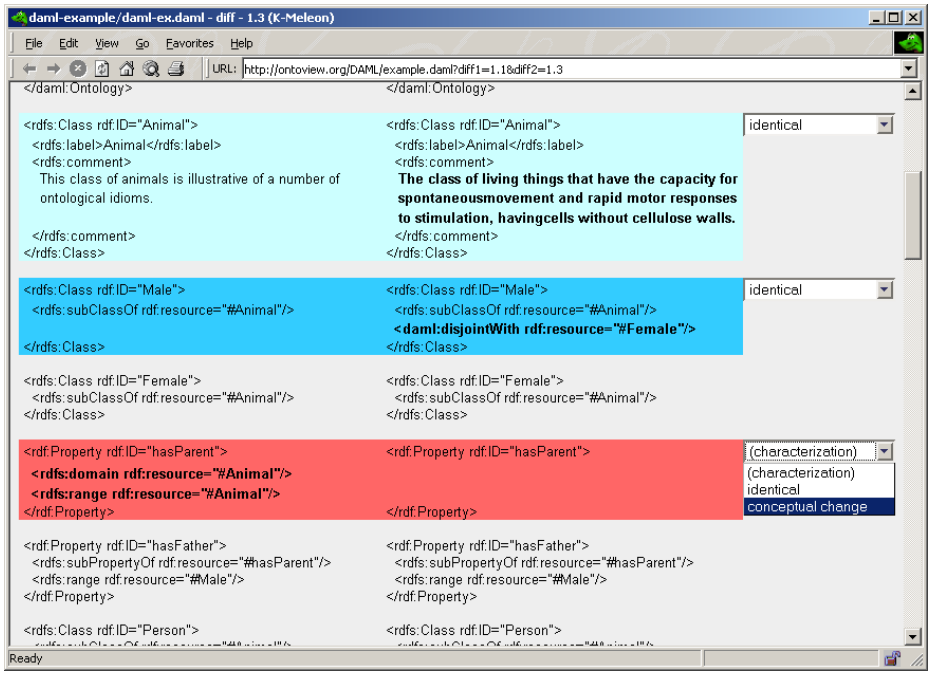

Fig. 2. Comparing two ontologies

cepts or properties are changed. An example of such a comparison of two versions of a DAML+OIL ontology is depicted in Figure 2.[7]

4.1 Types of change

The comparison function distinguishes between the following types of change:

- Non-logical change, e.g. in a natural language description. In DAML+OIL, this are changes in the rdfs:label of an concept or property, or in a comment inside a definition. An example is the first highlighted change in Figure 2 (class “Animal’).
- Logical definition change. This is a change in the logical definition of a concept. Examples of such changes are alterations of subClassOf, domain, or range statements. Additions or deletions of local property restrictions in a class are also logical changes. The second and third change in the figure is (class “Male” and property “hasParent”) are examples of such changes. Note that there are also logical changes that do not affect the semantics. However, we
- Identifier change. This is the case when a concept or property is given a new identifier, i.e. a renaming.
- Addition of definitions.

[7] This example is based on fictive changes to the DAML example ontology, available from `http://www.daml.org/2001/03/daml+oil-ex.daml`.

– Deletion of definitions.

Each type of change is highlighted in a different color, and the actually changed lines are printed in boldface.

Most of these changes can be detected completely automatically, except for the identifier change, because this change is not distinguishable from a subsequent deletion and addition of a simple definition. In this case, the system uses the location of the definition in the file as a heuristic to determine whether it is an identifier change or not.

It is a deliberate choice not to show all changes, but only the ones which we think that are of interest to the ontology modeler. This choice is explained in the next paragraphs, together with the mechanism that we use to detect and classify changes. Experimental validation should show whether this list of change types is sufficient.

4.2 Detecting changes

There are two main problems with the detection of changes in ontologies. The first problem is the abstraction level at which changes should be detected. Abstraction is necessary to distinguish between changes in the representation that affect the meaning, and those that don't influence the meaning. It is often possible to represent the same ontological definition in different ways. For example, in RDF Schema, there are several ways to define a class:

```
<rdfs:Class rdf:ID="ExampleClass"/>
```

or:

```
<rdf:Description rdf:ID="ExampleClass">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>
```

Both are valid ways to define a class and do not change the ontology. Thus, detecting changes in the *representation* alone is not sufficient.

However abstracting too far is also a problem: considering the *logical meaning* only is not enough. In [2] is shown that different sets of ontological definitions can yield the same set of logical axioms. Although the logical meaning is not changed in such cases, the ontology definitely is. Finding the right level of abstraction is thus important.

Second, even when we found the correct level of abstraction for change detection, the conceptual implication of such a change is not yet clear. Because of the difference between conceptual changes and explication changes (as described in section 2), it is not possible to derive the conceptual consequence of a change completely on basis of the visible change only (i.e., the changes in the definitions of concepts and properties). Heuristics can be used to suggest conceptual consequences, but the intention of the engineer determines the actual conceptual relation between versions of concepts.

In the next two sections, we explain the algorithm that we used to compare ontologies at the correct abstraction level, and how users can specify the conceptual implication of changes.

4.3 Rules for changes

The algorithm uses the fact that the RDF data model [10] underlies a number of popular ontology languages, including RDF Schema and DAML+OIL. The RDF data model basically consists of triples of the form <subject, predicate, object>, which can be linked by using the object of one triple as the subject of another. There are several syntaxes available for RDF statement, but they all boil down to the same data model. An set of related RDF statements can be represented as a graph with nodes and edges. For example, consider the following DAML+OIL definition of a class “Person”.

```
<daml:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

When interpreted as a DAML+OIL definition, it states that a “Person” is a kind of ”Animal” and that the instances of its “hasParent” relation should be of type “Person”. However, for our algorithm, we are first of all interested in the RDF interpretation of it. That is, we only look at the triples that are specified, ignoring the DAML+OIL meaning of the statements. Interpreted as RDF, the above definition results in the following set of triples:

subject	**predicate**	**object**
Person	rdf:type	daml:Class
Person	rdfs:subClassOf	Animal
Person	rdfs:subClassOf	*anon-resource*
anon-resource	rdf:type	daml:Restriction
anon-resource	daml:onProperty	hasParent
anon-resource	daml:toClass	Person

This triple set is depicted as a graph in Figure 3. In this figure, the nodes are resources that function as subject or object of statements, whereas the arrows represent properties.

The algorithm that we developed to detect changes is the following. We first split the document at the first level of the XML document. This groups the statements by their intended “definition”. The definitions are then parsed into RDF triples, which results in a set of small graphs. Each of these graphs represent a specific definition of a concept or a property, and each graph can be identified with the identifier of the concept or the property that it represents.

Then, we locate for each graph in the new version the corresponding graph in the previous version of the ontology. Those sets of graphs are then checked according to a number of rules. Those rules specify the “required” changes in the triples set (i.e., the graph) for a specific type of change, as described in section 4.1.

Rules have the following format:

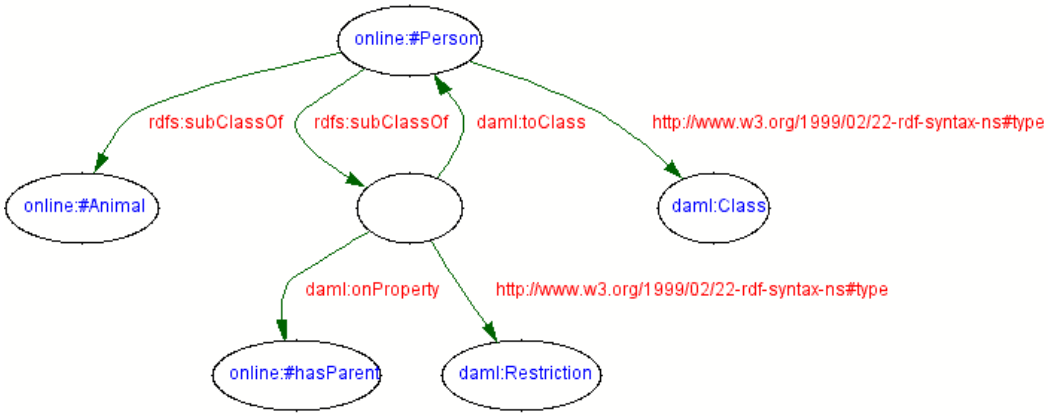

Fig. 3. An RDF graph of a DAML class definition.

```
IF exist:old
     <A, Y, Z>*
   exist:new
     <X, Y, Z>*
   not-exist:new
     <X, Y, Z>*
THEN change-type A
```

They specify a set of triples that should exists in one specific version, and a set that should not exists in another version to signal a specific type of change. With this rule mechanism, we were able to specify almost all types of change, except the identifier change. Here we also used some heuristics, based on the location of the definition in the file. We list two example rules below.

A change in the value of a local property:

```
IF exist:old
     <X, rdfs:subClassOf, Y1>
     <Y1, rdf:type, daml:#Restriction>
     <Y1, daml:onProperty, Y2>
     <Y1, daml:toClass, Z>
   exist:new
     <X, rdfs:subClassOf, Y1>
     <Y1, rdf:type, daml:#Restriction>
     <Y1, daml:onProperty, Y2>
   not-exist:new
     <Y1, daml:toClass, Z>
THEN logicalChange.localPropertyValue X
```

A change in the property type:

```
IF exist:old
```

```
     <X, rdf:type, rdf:#Property>
     <X, rdf:type, daml:#UniqueProperty>
   exist:new
     <X, rdf:type, rdf:#Property>
   not-exist:new
     <X, rdf:type, daml:#UniqueProperty>
THEN logicalChange.propertytype X
```

The rules are specialized for a specific RDF-based ontology language (in this case DAML+OIL), because they encode the interpretation of the semantics of the language for which they are intended. For another language, other rules would have been necessary to specify other differences in interpretation. The semantics of the language are thus encoded in the rules. For example, the last example not looks at changes in values of predicates (as the first does), but at a change in the type of property. This is a change that is related to the specific semantics of DAML+OIL.

In the prototype system, we were able to specify all changes types that we wanted to detect via this rule format.

4.4 Specifying the conceptual implication of changes

The comparison function also allows the user to *characterize* the conceptual implication of the changes. For the first three types of changes that were listed in section 4.1, the user is given the option to label them either as "identical" (i.e., the change is an explication change), or as "conceptual change", using the drop-down list next to the definition (Figure 2). In the latter case, the user can specify the conceptual relation between the two version of the concept. For example, the change in the definition of "hasParent" could by characterized with the relation $\texttt{hasParent}_{1.1}$ `subPropertyOf` $\texttt{hasParent}_{1.3}$.

5 Conclusion

Conceptual models will play an important role in the envisaged "Semantic Web". Versioning support for such ontologies is essential when they are used in such a distributed and dynamic context. In this paper we have analyzed the versioning relation and have described a system that provides support for change management of online ontologies.

In the system that we described, all the dimensions of a versioning relation are specified separately: the descriptive **meta-data**, the **conceptual relations** between constructs in the ontologies, and the **transformations** between the ontologies themselves. This allows both complete transformations of ontology representations and partial data reinterpretations. The conceptual relations can be exported and used to adapt data sources and ontologies.

We described how the systems support users in comparing ontologies, and what the problems and challenges are. We presented a algorithm to perform a comparison for RDFS-based ontologies. This algorithm doesn't operate on the representation of the ontology, but on the data model that is underlying the representation. By grouping the RDF-triples per definition, we still retained the necessary representational knowledge.

We also explained how users can specify the conceptual implication of changes to help interoperability. This honors the fact that it is not possible to derive all conceptual implications of changes automatically. An important advantage of this approach is that there is no write-access needed to the original ontologies. The exported conceptual relations between versions of concepts live on their own as separate mapping ontologies.

The described system is not yet finished and should be developed further. We believe that it will significantly simplify the change management of ontologies and thus help the interoperability of different domain models on the web.

References

1. J. Banerjee, W. Kim, H.-J. Kim, and H. F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD Record (Proc. Conf. on Management of Data)*, 16(3):311–322, May 1987.
2. S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not enough. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA, July 30 – Aug. 1, 2001.
3. S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris. A proposal for a description logic interface. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 33–36, Linköping, Sweden, July 30 – Aug. 1 1999.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.
5. D. Brickley and R. V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Wide Web Consortium, Mar. 2000.
6. D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In R. Dieng and O. Corby, editors, *Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International Conference EKAW 2000*, number 1937 in LNCS, pages 1–16, Juan-les-Pins, France, Oct. 2–6, 2000. Springer-Verlag.
7. D. Fensel and M. A. Musen. The semantic web: A new brain for humanity. *IEEE Intelligent Systems*, 16(2), 2001.
8. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 1993.
9. M. Klein and D. Fensel. Ontology versioning for the Semantic Web. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, pages 75 – 91, Stanford University, California, USA, July 30 – Aug. 1, 2001.
10. O. Lassila and R. R. Swick. Resource Description Framework (RDF): Model and Syntax Specification. Recommendation, World Wide Web Consortium, Feb. 1999. See http://www.w3.org/TR/REC-rdf-syntax/.
11. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 483–493, San Francisco, 2000. Morgan Kaufmann.
12. J. F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393, 1995.
13. P. R. S. Visser, D. M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. An analysis of ontological mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.