Finding and specifying relations between ontology versions

Michel Klein[1] and **Atanas Kiryakov**[2] and **Damyan Ognyanoff**[3] and **Dieter Fensel**[4]

Abstract. Interoperability between different existing ontologies is import to leverage the use of ontologies. However, the interoperability between different *versions* of ontologies is at least as important. Especially when ontologies are used in a distributed and dynamic context as the Web, we can expect that ontologies will rapidly evolve and thus may cause incompatibilities. This paper describes a system that helps to keep different versions of web-based ontologies interoperable. To achieve this, the system allows ontology engineers to compare versions of ontology and to specify the conceptual relations between the different versions of concepts. Internally, the system maintains the transformations between ontologies, some meta-data about the version, as well as the conceptual relation between concepts in different versions. This paper briefly describes the system, presents the mechanism that we used to find and classify changes in RDF-based ontologies, and discusses how this may be used to specify relations between ontologies that improve their interoperability.

1 ONTOLOGY EVOLUTION THREATENS INTEROPERABILITY

Ontologies have become popular because of their promise of knowledge sharing and reuse [10]. Interoperability between ontologies is an important issue, because the reuse of knowledge often implies that different existing ontologies are used together. This requires that the knowledge represented in the ontologies is not conflicting. However, ontology interoperability is not only important between different existing ontologies, it is also an issue between different *versions* of an ontology. This is especially relevant when ontologies are used in the context of the Semantic Web [5].

In this vision, ontologies have a role in defining and relating concepts that are used to describe data on the web. The distributed and dynamic character of the web will cause that many versions and variants of ontologies will arise. 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 [11].

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 goal of this system is not to provide a central registry for ontologies, but to allow ontology engineers to store their versions and variants of ontologies and relate them to other (possibly remote) ontologies. The resulting mapping relations between versions can also be exported and used outside the system.

Most of the ideas underlying the versioning 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[5] [8, 9] and RDF Schema (RDFS) [7] 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 an 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 6.

2 THE UPDATE RELATION BETWEEN ONTOLOGIES

There are three important aspects to discuss when considering an update relation between ontologies. 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. A change constitutes a new version of the ontology. This new version defines an orthogonal relation between the

[1] Vrije Universiteit Amsterdam, michel.klein@cs.vu.nl
[2] OntoText, Sofia, Atanas.Kiryakov@sirma.bg
[3] OntoText, Sofia, damyan@sirma.bg
[4] Vrije Universiteit Amsterdam, dieter@cs.vu.nl

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

definitions of concepts and properties in the original version of the ontology and those in the new version. This is depicted in Figure 1.

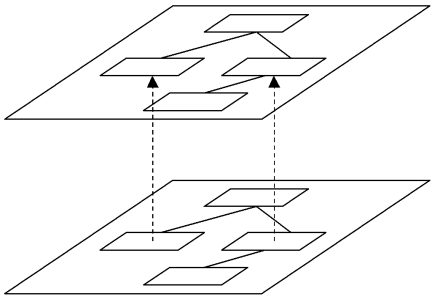

Figure 1. Orthogonal relations between classes in two version of an ontology (dashed arrows)

The relations between concepts inside an ontology, e.g. between class A and class B, is thus a fundamentally different relation from the update relation between two versions of a concept, e.g. between class $A_{1.0}$ and class $A_{2.0}$. In the first case, the relation is a purely conceptual relation in the domain; in the second case, however, the relation describes meta-information about the change of the concept.

Nevertheless, two version of a concept still have *some* conceptual relation. This relation, however, is not determined by the update itself, but accompanying information of an update relation. There are other characteristics of an update relation, too. We distinguish the following properties that can be associated with 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 [15] (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 [11], 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, and, the other way around, a concept can change without a change in its logical definition. An example of the first case is 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. On the other hand, a natural language definition of a concept might change, e.g. the new definition of "chair" might exclude reclining-chairs" without a logical change of the concept.

The intention of a change is made explicit by categorizing them into the following categories [16]:

- **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 change cannot be automatically 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.

A third, somewhat different, aspect of an update is the **packaging of changes**, i.e., the way in which updates are applied to an ontology. This is an important practical issue for the development of an ontology change management system.

We can distinguish two different dimensions with respect to the packaging of the change specification. One dimension is the *granularity* of the specification: this can be either the level of a single "definition" or the level of a "file" as a whole.

The second dimension is the *method* of specification. There are several methods thinkable:

- a "transformation specification": an update specified by a list of change operations (e.g., add A, change B, delete C);
- a "replacement": an update specified by replacing the old version of a concept or an ontology with a new version; this is an implicit change specification;
- a "mapping": an update specified as a mapping between the original ontology and another one. Although this is not a update in the regular sense, an explicit mapping to another ontology can be considered as an update to the viewpoint of that ontology.

This gives several possible change specifications. For example, a change can be specified individually, as a mapping between one specific definition in one ontology and another definition in another ontology, but it can also be done at a file level, by defining the transformation of the ontology.

Notice that the packaging methods are not equivalent, i.e., they do not give the same information about the update relation. It is clear that the mapping provides a conceptual relation between versions of concepts that is not specified in a transformation.

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.

It also provides 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 [4], 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[6]. 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[7].

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

- **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 first case, the user also has to provide a "location" for the ontology in the hierarchical structure of the OntoView system.
  Then, the user is guided through a short process in which he is asked to supply the meta-data of the version (as far as this can not be derived automatically, such as the date and user), to characterize the types of the changes (see below in section 4), and to decide about the identifier of the 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.
  Usually, the XML Namespace mechanism [6] is used for the identification of web-based ontologies. This means that an ontology is identified by a URI, i.e. a unique pointer on the web. In practice, people tend to use the location (the URL) of the ontology file on the web as identifier. OntoView also uses the namespace mechanism for identification, but does not necessarily use the location of the ontology file. If a change does not constitute a conceptual change, the new version gets a new location, but does not get a new identifier. For example, the location of an ontology can change from "../example/1.0/rev0" to "../example/1.0/rev1", while the identifier is still "../example/1.0".
  OntoView supports two ways of persistent and unique identification of web-based ontologies. First, it can in itself guarantee the uniqueness and persistency of namespaces that start with "http://ontoview.org/", because the system is located at the domain `ontoview.org`. Second, because the location and identification of ontologies are only loosely coupled, it can also store ontologies with arbitrary namespaces. In this case, the ontology engineer is responsible for guaranteeing the uniqueness. The ontologies with arbitrary namespaces are not directly retrievable by their namespace, but can be accessed via a search function.
- **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 [13].
  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.
  The exported mapping ontologies are represented with the standard constructs of the ontology langauge. Because in OntoView the conceptual relation and the actual transformation are stored separately, it is not necessary to extend the ontology language with more advanced mapping- or transformation primitives than those already available.
  The meta-data about the ontology update is specified as a set of properties of the conceptual relations themselves. In DAML+OIL, this meant that we had to re-ify the mapping statements.[8] This method has two advantages. First, when specified over re-ified statements, the meta-data does not interfere with the actual ontological knowledge, as would be the case when meta-data is specified as characteristics of classes and properties. Second, because the meta-data is data about the *mappings themselves*, agents or systems that understand the meta-data can use this to decide which mappings are applicable in a specific context and which are not.
  In the future, it should also be possible to export *transformations* between two versions of an ontology. A transformation is a complete specification of all the change operations. This can be used to re-execute changes and to update ontologies that have some overlap with the versioned ontology in exactly the same way as the original one. However, transformations facilitates data re-interpretations only to a very small extent. A mapping ontology provides better re-interpretation, because it also captures human knowledge about the relations.

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

[6] Available from `http://stud.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/`

[7] A demo is available at `http://sesame.aidministrator.nl`

[8] The DAML+OIL semantics do not currently cover reification because of the undecidability of second-order logic. However, there is an awareness that use reification for "tagging" purposes — as we do — is different from full second-order logic. See `http://www.daml.org/language/features.html`.

textually differ in two versions. OntoView, in contrast, compares version of ontologies at a *structural* level, showing which definitions of ontological concepts or properties are changed. An example of such a graphical comparison of two versions of a DAML+OIL ontology is depicted in Figure 2.[9]

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 definition of a concept that affects its formal semantics. 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.
- Identifier change. This is the case when a concept or property is given a new identifier, i.e. a renaming.
- Addition of definitions.
- Deletion of definitions.

Most of these changes can be detected completely automatically, except for the identifier change. Each type of change is highlighted in a different color, and the actually changed lines are printed in boldface. We describe the mechanism that we use to detect and classify changes in the next paragraphs.

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="...chema#Class"/>
</rdf:Description>
```

Both are valid ways to define a class and have exactly the same meaning. Such a change in the representation would not change the ontology. Thus, detecting changes in the *representation* alone is not sufficient.

However abstracting too far can also be 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 [12] 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.

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

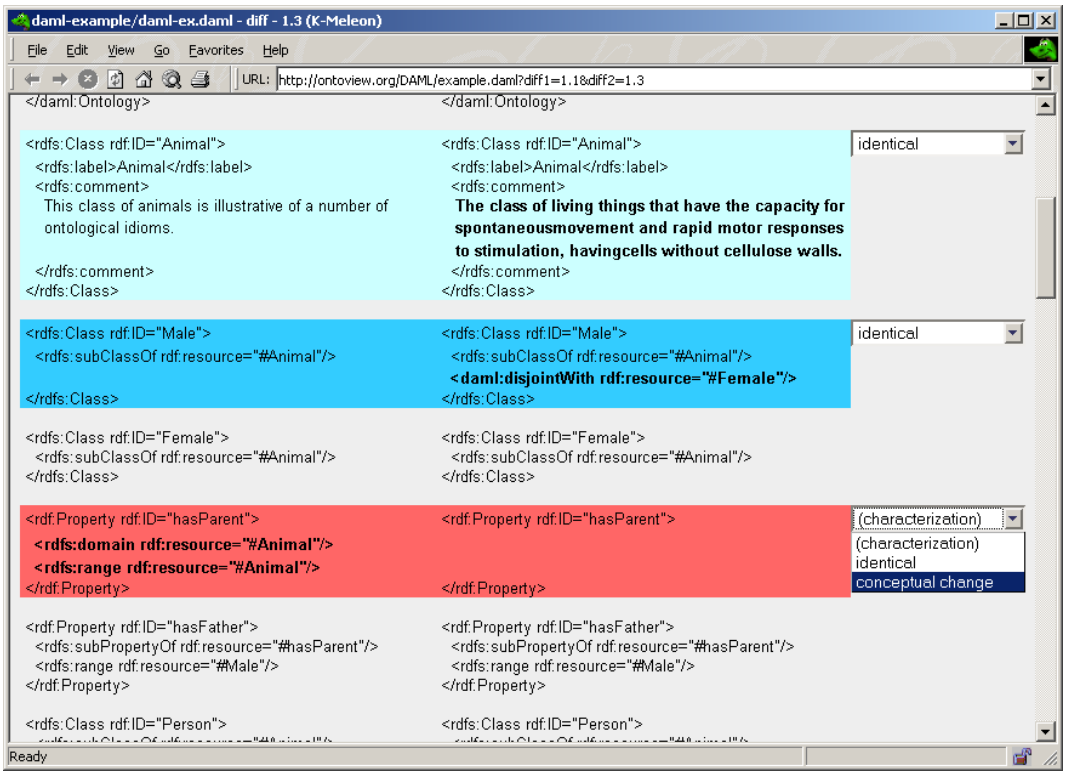

Figure 2. Comparing two ontologies

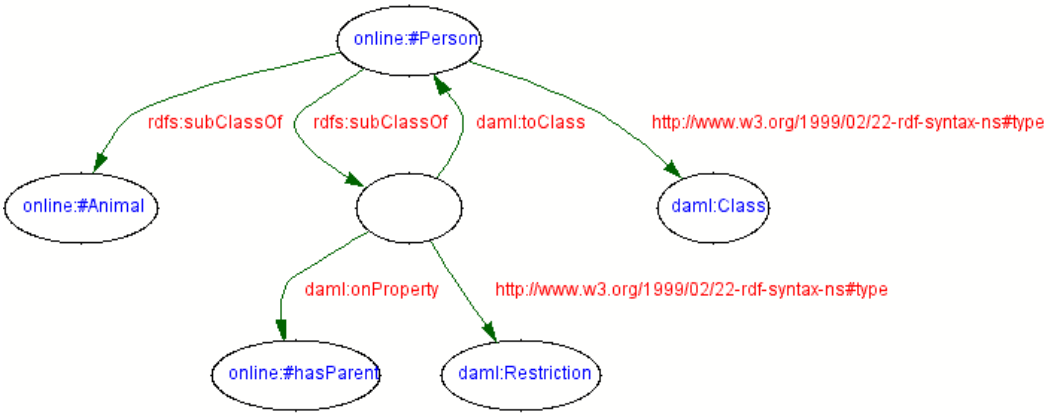

Figure 3. An RDF graph of a DAML class definition.

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:

```
IF exist:old
     <A, 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 (or the other way around) 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>
   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>
   not-exist:new
     <X, rdf:type, daml:#UniqueProperty>
THEN logicalChange.propertytype X
```

The rules are specific for a particular 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.

Also, notice that the mechanism relies on the "materialization" of all `rdf:type` statements that are encoded in the ontology (sometimes called "knowledge compilation"). The last example depend on the existence of a statement `<X,rdf:type,rdf:#Property>`. However, this statement can only be derived using the semantics of the `rdfs:subPropertyOf` statement, which — informally spoken[10] — says that if a property is an instance of type X, then it is also an instance of the supertypes of X. The application of the rules thus has to be preceded by the materialization of the superclass- and superproperty hierarchies in the ontology. For this materialization, the entailment rules in the RDF Model Theory[11] can be used.

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 DISCUSSION

There are a few other issues and choices about the design of the system that we want to discuss. First, we purposely do not provide support for finding mappings between arbitrary ontologies. The intention of our system is to provide users with a system to manage versions of ontologies and maintain their relations. Finding the relations is a different task. However, it might be possible to incorporate this function in a future version of the system, e.g. by interfacing it with a ontology mapping tool.

Another issue is the visualization of the changes. The current versions shows the changes by highlighting the textual definitions that are changed. More advanced visualization techniques are possible. For example, one could think of techniques that render ontologies in a graphical representation and highlight the changes in the picture.

We did not yet specify the way in which a "valid context" is described. Such a context will have several dimensions, of which "time" is only one. This is something what still has to be done. Without such a specification, it is difficult to assess the validness of a conceptual relation between concepts in different versions. We can assume that such a relation is at least valid between two successive versions, but we do not know whether such mapping is allowed to "propagate" via other mappings to other ontologies. Research on this is necessary.

A situation in which versioning support is also necessary is the collaborative development of an ontology [14]. We think that OntoView is also useful in this situation, especially because all the conceptual implications of versions have to be characterized individually by users. This integrates the conflict resolution in the update procedure.

A side remark about the use of a versioning system for collaborative ontology development is that this gives an evolutionary way of ontology building. Each person can have its own conceptualization, which is conceptually linked to the conceptualizations of others. In this sense, the combination of versions and adaptations in itself forms a *shared* conceptualization of a domain.

Finally, we want to mention that the system is still under construction. In section 3 we extensively depicted the foreseen functionality of OntoView. However, as became clear of some of the descriptions, not everything is already realized. The basis functions are implemented, but a number of more advanced functions are still being developed.

6 CONCLUSION

When ontologies are used in a distributed and dynamic context, versioning support is essential ingredient to maintain interoperability. In this paper we have analyzed the versioning relation, described its aspects, and depicted a system that provides support for the versioning of online ontologies.

We described how this systems supports helps users to compare ontologies, and what the problems and challenges are. We presented a algorithm to perform a comparison for RDF-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.

The analysis of a versioning relation between ontologies revealed several dimensions of it. In the system that we described, all these dimensions are maintained separately: the descriptive **meta-data**, the **conceptual relations** between constructs in the ontologies, and the **transformations** between the ontologies themselves. This multi-dimensional specification allows both complete transformations of ontology representations and partial data re-interpretations, which help interoperability. The conceptual differences can be exported and used stand alone, for example to adapt data sources and 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 evolving ontologies on the web.

[10] The precise semantics of RDF Schema are still under discussion.
[11] `http://www.w3.org/TR/rdf-mt/`

REFERENCES

[1] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry 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] Sean Bechhofer, Carole Goble, and Ian Horrocks, 'DAML+OIL is not enough', in *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA, (July 30 – August 1, 2001).

[3] S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris, 'A proposal for a description logic interface', in *Proceedings of the International Workshop on Description Logics (DL'99)*, eds., P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, pp. 33–36, Linköping, Sweden, (July 30 – August 1 1999).

[4] Brian Berliner, 'CVS II: Parallelizing software development', in *Proceedings of the Winter 1990 USENIX Conference*, ed., USENIX Association, pp. 341–352, Washington, DC, USA, (January 22–26, 1990). USENIX.

[5] Tim Berners-Lee, Jim Hendler, and Ora Lassila, 'The semantic web', *Scientific American*, **284**(5), 34–43, (May 2001).

[6] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. `http://www.w3.org/TR/REC-xml-names/`, January 1999.

[7] D. Brickley and R. V. Guha, 'Resource Description Framework (RDF) Schema Specification 1.0', Candidate recommendation, World Wide Web Consortium, (March 2000).

[8] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Stefan Decker, Michael Erdmann, and Michel Klein, 'OIL in a nutshell', in *Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International Conference EKAW 2000*, eds., Rose Dieng and Olivier Corby, number LNCS 1937 in Lecture Notes in Artificial Intelligence, pp. 1–16, Juan-les-Pins, France, (October 2–6, 2000). Springer-Verlag.

[9] Dieter Fensel and Mark A. Musen, 'The semantic web: A new brain for humanity', *IEEE Intelligent Systems*, **16**(2), (2001).

[10] T. R. Gruber, 'A translation approach to portable ontology specifications', *Knowledge Acquisition*, **5**(2), (1993).

[11] Michel Klein and Dieter Fensel, 'Ontology versioning for the Semantic Web', in *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA, (July 30 – August 1, 2001).

[12] O. Lassila and R. R. Swick, 'Resource Description Framework (RDF): Model and Syntax Specification', Recommendation, World Wide Web Consortium, (February 1999). See http://www.w3.org/TR/REC-rdf-syntax/.

[13] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder, 'An environment for merging and testing large ontologies', in *KR2000: Principles of Knowledge Representation and Reasoning*, eds., Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, pp. 483–493, San Francisco, (2000). Morgan Kaufmann.

[14] Helena Sofia Pinto and Jo ao Pavão Martins, 'Evolving ontologies in distributed and dynamic settings', in *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, Toulouse, France, (April 22–25, 2002).

[15] John F. Roddick, 'A survey of schema versioning issues for database systems', *Information and Software Technology*, **37**(7), 383–393, (1995).

[16] Pepijn R. S. Visser, Dean 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).