Ontology versioning on the Semantic Web

Michel Klein and Dieter Fensel
Vrije Universiteit Amsterdam
De Boelelaan 1081a
1081 HV Amsterdam, the Netherlands
michel.klein/dieter@cs.vu.nl

Abstract Ontologies are often seen as basic building blocks for the Semantic Web, as they provide a reusable piece of knowledge about a specific domain. However, those pieces of knowledge are not static, but evolve over time. Domain changes, adaptations to different tasks, or changes in the conceptualization require modifications of the ontology. The evolution of ontologies causes operability problems, which will hamper their effective reuse. A versioning mechanism might help to reduce those problems, as it will make the relations between different revisions of an ontology explicit. This paper will discuss the problem of ontology versioning. Inspired by the work done in database schema versioning and program interface versioning, it will also propose building blocks for the most important aspects of a versioning mechanism, i.e., ontology identification and change specification.

1 Introduction

Ontologies are often seen as basic building blocks for the Semantic Web, as they provide a reusable piece of knowledge about a specific domain. However, those pieces of knowledge are often not static, but evolve over time. Domain changes, adaptations to different tasks, or changes in the conceptualization require modifications of the ontology. The evolution of ontologies causes operability problems, which will hamper the effective reuse.

Support to handle those changes is needed. This is especially important in a decentralized and uncontrolled environment like the web, where changes occur without attunement. Much more than in an controlled environment, this may have unexpected and unknown results. With the rise of the Semantic Web, those uncontrolled changes will have even more impact, because *computers* will use the data. There are no longer humans in the chain that — using a vast amount of background knowledge and implicit heuristics — can spot erroneous combinations due to unexpected changes.

The problem is even worse, because there are a lot of dependencies between data sources, applications and the ontologies. Changes to the latter will thus have far-reaching side effects. It is often not practically possible to synchronism the changes to an ontology with modifications to the applications and data sources that use them. Therefore, a versioning methodology is needed to handle revisions of ontologies and the impact on existing sources.

In this paper, we will explore the problem of ontology versioning and we will propose some elements of a versioning framework. We will first discuss the nature of ontology versioning in Section 2. Because compatibility is a key issue in versioning, Section 3 contains an

analysis of the compatibility between schema's and conforming data. To come up with concrete requirements for a versioning framework, we will look at current practices for handling ontology change in Section 4, in which we will also formulate those requirements. Section 5 will eventually present the proposed baseline for an ontology versioning framework on the web. It will mainly concentrate on identification and referring issues. Section 6 concludes the paper and sketches the directions for further research. Finally, the appendix shows how this ideas may be implemented in RDF Schema and / or DAML+OIL.

2 The problem: versioning of ontologies

Before we can investigate the solutions for ontology versioning, we first have to take a closer look at what it actually is. In a general sense, ontology versioning just means that there are multiple variants of an ontology around. In practice, those variants often originates from changes to an existing variant of the ontology and thus form a derivation tree. It is also possible that different "versions" of ontologies of the same domain are independently developed and do *not* have a derivation relation. In this case, however, we will not use the word "version", but we will see the variants as separate ontologies that describe the domain from a specific *viewpoint* or *perspective*. In our view, ontology versioning is closely related to changes in ontologies.

We define ontology versioning as *the ability to handle changes in ontologies by creating and managing different variants of it*. To achieve this ability, we need a methodology with methods to distinguish and recognize versions, and with procedures for updates and changes in ontologies. This also implies that we should keep track of the relationships between versions.

Focused on ontologies, we can say that a *versioning methodology* provides mechanism to disambiguate the interpretation of concepts for users of the ontology variants, and that it makes the compatibility of the variants explicit. The extend of the changes determines the compatibility between the versions. This implies that the semantic impact of changes should be examined. The central question that a versioning methodology answers is: how to reuse existing ontologies in new situations, without invalidating the current usage.

2.1 Causes of ontology changes

A versioning methodology for ontologies copes with changes in *ontologies*. To examine the causes of changes, we will have to look at the nature of ontologies. According to Gruber (1993), an ontologies is a *specification of a shared conceptualization of a domain*. Hence, changes in ontologies are caused by either:

1. changes in the domain;
2. changes in the shared conceptualization;
3. changes in the specification.

The first type of change is often occurring. This problem is very well known from the area of database schema versioning. Ventrone and Heiler (1991) sketches seven different situations in which changes in a domain (domain evolution) require changes to a database model. An

example of this type of change is the merge of two university departments: this is a change in the real world, which requires that an ontology that describes this domain is modified, too.

Changes in the shared conceptualization are also frequently happening. It is important to realize that a *shared* conceptualization of a domain is not a static specification that is produced once in the history, but has to be reached over time. Fensel (2001) describes ontologies as dynamic networks of meaning, in which consensus is achieved in a social process of exchanging information and meaning. This view attributes a dual role to ontologies in information exchange: they provide consensus that is both a *pre-requisite* for information exchange and a *result* of this exchange process.

A conceptualization can also change because of the usage perspective. Different tasks may imply different views on the domain and consequently a different conceptualization. When an ontology is adapted for a new task or a new domain, the modifications represent changes to the conceptualization. For example, consider an ontology about traffic connections in Amsterdam, with concepts like roads, cycle-tracks, canals, bridges and so on. When the ontology is adapted from a bicycle perspective to a water transport perspective, the conceptualization of a bridge changes from a remedy for crossing a canal to a time consuming obstacle[1].

Finally, a specification change is a kind of translation, i.e., a change in the way in which a conceptualization is formally recorded. Although ontology translation is an important and non-trivial issue in many practical applications, it is less interesting *from a versioning perspective*, for two reasons. First, an important goal of a translation is to retain the semantics, i.e., specification variants should be equivalent[2] and they thus only cause syntactic interoperability problems. Second, a translation is often created to use the ontology in an other context (i.e., an other application or system), which heavily reduces the importance of interoperability questions. Therefore, we will leave specification changes alone and concentrate on support for changes in the semantics of an ontology, caused by either domain changes or conceptualization changes.

2.2 Consequences of the change

Versioning support is necessary because changes to ontologies may cause incompatibilities, which means that the changed ontology can not simply be used instead of the unchanged version. There are several type of things that may depend on an ontology. Each of these dependencies may cause a different type of incompatibility.

- In the first place, there is the data that conforms to the ontology. In a semantic web, this can be web pages of which the content is annotated with terms from an ontology. When an ontology is changed, this data may get an different interpretation or may use unknown terms.

- Second, there are other ontologies that use the changed ontology. This may be ontologies that are built from the source ontology, or that import the ontology. Changes to the source ontology may affect the resulting ontology.

[1] Actually, for many people this meaning is also an element of the bicycle perspective.

[2] Although in practice a translation often implies a change in semantics, possibly caused by differences in the representation languages. See for a exploration of ontology language differences and mismatches (Corcho and Gómez-Pérez, 2000) and (Klein, 2001).

- Third, applications that use the ontology may also be hampered by changes to the ontology. In the ideal case, the conceptual knowledge that is necessary for an application should be merely specified in the ontology; however, in practice applications also use an internal model. This internal model may become incompatible with the ontology.

All things considered, we see that a versioning methodology is necessary to take care of the following relations:

- between succeeding revisions of one ontology;
- between the ontology and:
  - instance data;
  - related ontologies;
  - related applications.

In the rest of the paper, we will mainly concentrate on the relation between *data sources* and related ontologies. This has two reasons. First, this specific dependency is occurring very frequently at the Semantic Web and therefore forms an urgent problem. Second, because the other dependencies can be considered as more complex cases of the first dependency, we will have to start with the first case to come to the more complex situations in future work.

3 Analysis of compatibility

In Section 2, we discussed the relations between ontologies and depending data sources in general. In this section, we will take a closer look at the compatibility of changed ontologies and data sources.

We can imagine that — when the Semantic Web evolves — there will be various versions of many ontologies around. There will also be a lot of web pages and applications that use (or are intended to use) a specific version of the ontology. Consequently, there are a number of (possible incompatibility) ways of combining versions of ontologies with versions of data sources. Basically, there are three ways to relate ontology versions with data sources: (1) using the intended version of the ontology for a data source, (2) using a newer version of the ontology, and (3) using an older version of the ontology. As the first type of of combination is naturally compatible, we thus have to explore the incompatibility of ontologies and data sources in two directions, which are illustrated in Figure 1. The terms that we use are known from the database schema versioning literature Roddick (1995).

- **prospective use**: the use of data sources that conform to a previous version of the ontology via a newer version of the ontology (*i.e. view the data from a newer perspective*).

- **retrospective use**: the use of data sources that conform to a newer version of the ontology via a previous version of the ontology (*view the data from an older perspective*).

Based on these directions of use, we can now categorize the compatibility of ontology revisions into different types. In this categorization, we examine whether it is valid to use a revision of an ontology on the *set of all possible instances* of an other revision of the ontology. In other words, we assume that the data source that conforms to a specific version of the ontology uses the whole ontology, i.e., all concepts and relations. From a compatibility perspective, this is a worst case scenario. Table 1 shows the types of compatibility.
We can describe the types of revisions as follows:

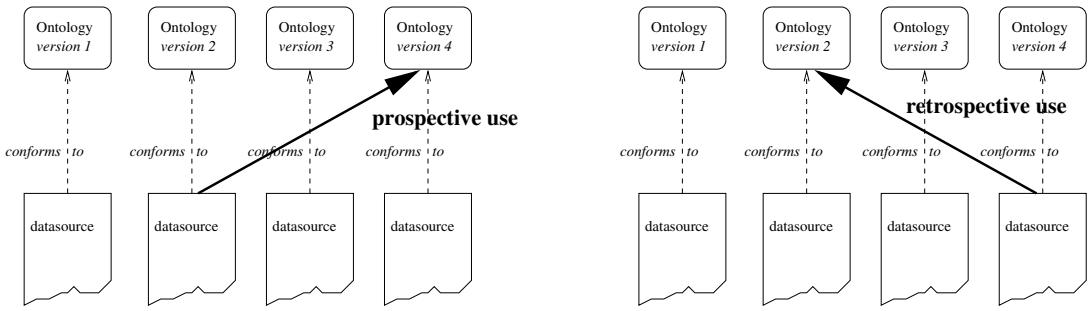

Figure 1: Two examples of prospective and retrospective use of ontologies.

		prospective use valid	
		yes	no
retrospective use valid	yes	full compatible	upward compatible
	no	backward compatible	incompatible

Table 1: Categorization of compatibility

- **full compatible revisions** (upward and backward): the semantics of the ontology is not changed, e.g. syntactic changes or updates of natural language descriptions; this type of change is compatible in both *prospective* use and *retrospective* use.

- **backward compatible revisions**: the semantics of the ontology are changed in such a way that the interpretation of data via the new ontology is the same as when using the previous version of the ontology, e.g. the addition of an independent class; this type of change is compatible in *prospective* use;

- **upward compatible revisions**: the semantics of the ontology is changed is such a way that an older version can be used to interpret newer data sources correctly, e.g. the removal of an independent class; this revision is compatible in *retrospective* use.

- **incompatible revisions**: the semantics of the ontology is changed in such a way that the interpretation of old data sources is invalid, e.g. changing the place in the hierarchy of a class; this type of change is incompatible in both *prospective* use and *retrospective* use.

Notice that both backward compatibility and upward compatibility are transitive: when the changes from $v1$ to $v2$ as well as the changes from $v2$ to $v3$ are backward compatible, then the changes from $v1$ to $v3$ are also backward compatible.

Consequently, if we know that all subsequent *revisions* to an ontology up to a certain version are backward compatible, it is also possible to name the resulting version of the *ontology itself* backward compatible. However, it is never allowed to call a version of an ontology *upward or full compatible*, because the semantics of future versions are not known beforehand! It is always possible that new versions of ontologies will introduce new things that cannot correctly be interpreted via older ontology versions. Thus, *backward compatibility* can be a characteristic of an ontology, but *upward or full compatibility* can not.

In the characterization above, we looked at compatibility from a theoretic perspective, considering data sources that consist of all possible instances of the ontology to which they

conform. As we already said, this is a worst case scenario. In practice, much more combinations of ontology versions and data sources yield a valid interpretation of the data than can be assumed from the schema in Table 1.

For example, although the table depicts that an incompatible revision can not be used for prospective interpretation (to interpret data sources that conforms to an older version), it is very well possible that a specific prospective use is valid, i.e., that the ontology can be used to correctly interpret some data that is committed to a previous version. This occurs when the *ontological commitment* of a data source, i.e., that part of the ontology that is actually "used" by a data source[3], is a subset of the complete "ontology" that is not affected by the revisions.

It is very much required that versioning methodologies and techniques for the Semantic Web exploit this "practical" compatibility where possible. That is, the techniques should not only determine whether a specific combination an ontology version and a data source version provides a valid interpretation *in general*, but — even when the combination is not generally valid — try to use as much knowledge as possible. We think that such techniques for "maximal exploitation" are essential for the development of the Semantic Web.

4 Current practices and requirements

To come up with concrete requirements for an ontology versioning mechanism on the web, we will now look at the current practices for managing changes in web-ontologies. We will describe a few typical scenarios for ontology change and explore the effects of those scenarios on two examples.

Currently, there is no agreed versioning methodology for ontologies on the web. However, in an decentralized and uncontrolled environment like the web, changes are certainly needed and do occur! When we look at the current practices, we can sketch several scenarios for ontology changes.

1. The ontology is silently changed; the previous version is replaced by the new version without any (formal) notification.

2. The ontology is visibly changed, but only the new version is accessible; the previous version is replaced by the new version.

3. The ontology is visibly changed, and both the new version and the previous version are accessible.

4. The ontology is visibly changed, both the new version and the previous version are accessible, and there is an explicit specification of the relation between concepts of the new version and the previous version.

4.1 Simple example

To come to concrete requirements for a versioning methodology, we will now look at the effects of these scenarios on the compatibility when an ontology changes. As an example we use an ontology of the education system in the Netherlands and web pages that are annotated with this ontology.

[3] This definition and the use of the term "ontology" is quite sloppy.

In the distant past, there was only one type of higher education, which was called an "University". A small part of an ontology that describes this looks as follows:[4]

class-def Education
class-def AcadamicEducation
  subclass-of Education
class-def Higher-Education-Institute
class-def University
  subclass-of Higher-Education-Institute
  slot-constraint *type-of-education*
    has-value AcadamicEducation

Many years later, a new type of higher-education was introduced, which provides professional education. This type was called "HBO". The above ontology has to be extended with the following three definitions. This addition is a monotonic extension to the ontology and the new version can be considered as being backward compatible with the first version.

class-def ProfessionalEducation
  subclass-of Education
disjoint AcadamicEducation ProfessionalEducation
class-def HBO
  subclass-of Higher-Education-Institute
  slot-constraint *type-of-education*
    has-value ProfessionalEducation

Let us suppose that there are a lot of web pages about education in the Netherlands around, which are annotated using the first version of the ontology. We will now try to interpret this information using the second version of the ontology (prospective use). We describe the effects of the change for each of the scenarios that are listed above.

Ex. 1, ad 1 When we have completely no clue that the ontology is changed, we encounter — in this backward compatible case — no problems at all. The terms that are used in the data source are the same as those in the ontology, and they also have the same meaning. A "University" on a web page is correctly interpreted.

Ex. 1, ad 2 When we know that the ontology is changed, but we don't know anything about the previous version of the ontology, nothing is sure anymore! Definitions could be changed and we can not derive that the term "University" on a web page (using ontology version 1) is the same as our definition.

Ex. 1, ad 3 In case the previous ontology can be accessed, we can compare and relate the ontologies, and see whether the changes interferes with the semantics of existing terms. In this case, we could have derived that the concept of "University" is not changed, and that the data sources can still correctly be interpreted.

[4]We use the "presentation syntax" of OIL (Fensel et al., 2000a) to represent the ontology; the interpretation is more or less straightforward. We could also have used DAML+OIL, but this would have required much more space.

Ex. 1, ad 4 If the relation between the concepts is explicitly specified, it would be clear that the the new version is backward compatible with the previous version, because the new version only adds a concepts. We could then safely conclude that the interpretation of previous data is still valid.

Notice that in the last three scenarios it is important to know which version of the ontology is used to annotate the data sources. This should me made explicit in some way.

4.2 More complicated example

In the year 2000, the Dutch government decided that both professional and academic institutes for higher-education are allowed to call themselves "University". This implies a new change to our ontology, resulting in a third version. This version is in general incompatible with the previous version. In the new version, the definition of "University" is changed to:

class-def University
  subclass-of Higher-Education-Institute
  slot-constraint *type-of-education*
    has-value (ProfessionalEducation **or** AcadamicEducation)

Let us again look at the consequences of this change with current versioning practices:

Ex. 2, ad 1 When we do not know that the ontology is changed, we use an other interpretation of a "University" than intended. Because in this case, a University-v1 is a subclass of University-v2, the interpretation of data is not incorrect, but also not complete. We cannot interpret that every "University" in the data sources actually provide academic education.

Ex. 2, ad 2 Same problem as with the change in the previous example.

Ex. 2, ad 3 If we have both version of the ontologies, we could compare them and see that only the definition of "University" is changed. We can see that both versions are subclasses of "Higher-Education-Institute", and interpret all instances of the old "University" as instance of "Higher-Education-Institute". This is again correct but incomplete. It ignores some knowledge that is available. Notice that, in this case, smart agents (agents capable of performing OIL classification) can derive that both Universities and HBOs are subclasses of new universities. Figure 2 shows[5] the classes in our example before and after classification.

Ex. 2, ad 4 In the case in which the relations between the concepts in the two ontologies are explicitly specified, it would tell us that "University" in the new ontology subsumes both "HBO" and "University" in the previous version.

It is also worth to notice that, in the last two scenarios, it is necessary to be able to distinguish between the different version of a concept. As the definition of "University" is changed, we need a separate identifier for each version of the definition. Otherwise, it is not possible to relate the previous and new definition to each other. In Figure 2, this is temporarily solved by appending "-v2" to the concept name.

[5] Modeled with the OILed tool, `http://img.cs.man.ac.uk/oil/`.

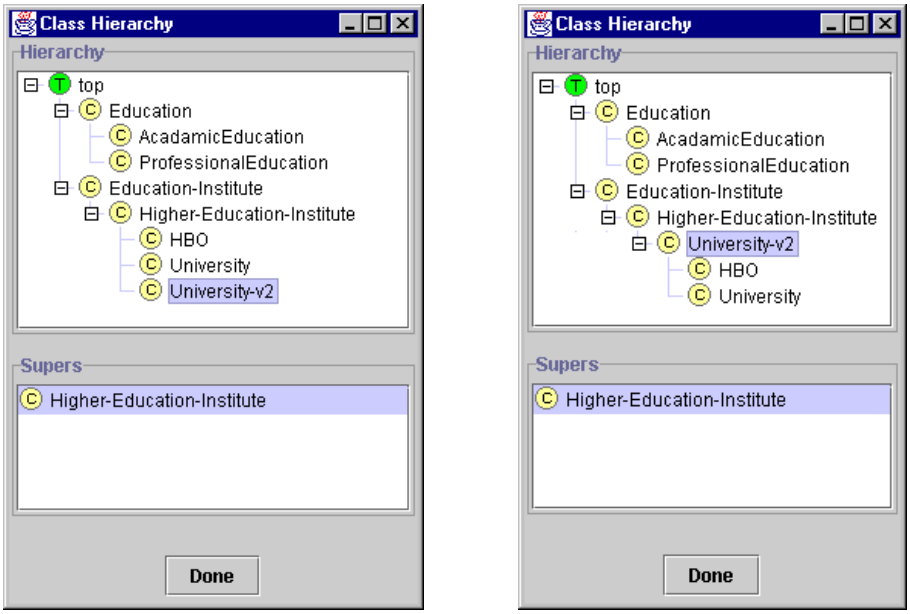

Figure 2: The hierarchy of the example ontology before and after classification with FaCT.

4.3 Observations

Based on the examples above, we can make a few observations. First, changing an ontology without any notification *may* result in a correct interpretation of the data. This is the case when the modification in the ontology does not affect the existing definitions, i.e., when the change is a monotonic extension. Heflin and Hendler (2000) show that the addition of concepts or relations are such extensions. When used on a data source, ontologies that are extended in this way yield the same perspective as when the original ontology is used. The interpretation is also valid when the revised concepts subsumes the original concepts. However, although the interpretation is correct in this case, it is only partial: not all the knowledge is exploited.

Because many changes in ontologies consist just of additions of concepts, it is understandable that the first scenario of ontology change is sometimes used. It is, however, not difficult to think of an change that — in this scenario — will result in an *invalid* interpretation, e.g., every change that restricts the extension of a class. This scenario should therefore be prevented.

Second, we see that it can be beneficial to have access to older versions of the ontology. This allows to compare the definitions and judge the validness of definitions used with other versions the data. In case of a cleanly modeled ontology, [6] it is even possible to have some automate support for this, e.g. by using the FaCT classifier[7].

As a third observation, we see knowledge about the relation concepts of different versions may yield in a partial but correct interpretation of the data. This relation can either be

[6]That is, the definitions of concepts should state whether they are necessary or necessary and sufficient; in Description Logic parlance: primitive or defined. This way of modeling is more naturally in OIL than in DAML+OIL, as the second requires that a defined concept is modeled as an equivalence to a couple of restrictions. It is therefore questionable whether in practice DAML+OIL ontologies can benefit much from the classification support. See also the discussion on this topic on the RDF-Logic mailing-list: `http://lists.w3.org/Archives/Public/www-rdf-logic/2001Mar/0000.html`.

[7]`http://www.cs.man.ac.uk/~horrocks/FaCT/`

manually specified, or can partly be derived, as described in the previous paragraph. For the specification of the relation between concepts of different versions, we need a identification mechanism for individual concepts of an ontology version. If we cannot refer to a specific version of a concept, this specification is not possible.

4.4 Requirements for versioning framework

Now we have explored the problems around ontology versioning, we can formulate requirements and wishes for a versioning methodology. We will first define the general goal for a versioning framework.

> A versioning methodology should provide mechanisms and techniques to manage changes to ontologies, while achieving maximal interoperability with existing data and applications. This means that it should retain as much information and knowledge as possible, without deriving incorrect information.

Notice that this is much stricter than just specifying whether it is valid to use a certain version of an ontology with a data source.

The general goal can be detailed in a number of more specific requirements. We impose the following requirements on a versioning framework, in increasing level of difficulty:

- for every use of a concept or a relation, a versioning framework should provide an unambiguous reference to the intended definition; (**identification**)
- a versioning framework should make the relation of one version of a concept or relation to other versions of that construct explicit; (**change specification**)
- a versioning framework should — as far as possible — automatically translate and relate the versions and data sources, to enable transparent access. (**transparent evolution**)

5 Building blocks for a versioning methodology

After we have stated the problems and requirements, we will now provide elements of a versioning methodology. We will concentrate our discussion on ontology identification and change specification.

5.1 Ontology identification on the web

Identity of ontologies The first question that has to be answered when we want to identify versions of an ontology on the web is: what is the identity of an ontology? This is not as trivial as it seems. In Section 3, we already anticipated this question by stating that syntactic changes and updates of natural language descriptions are fully compatible revisions. However, this is very debatable! If an ontology is seen as a specification of a conceptualization, then every modification to that specification can be considered a new conceptualization of the domain. In that case, the descriptions specify different concepts, which are *per definition* not equal.

Looking at this from another perspective, one might regard an ontology primarily as a conceptualization, which is represented as complete as possible in a specification. In this

case one could argue that an update to a natural language description of a concept is not a semantic change, but just a refined description of the same conceptualization.

In this philosophical debate, we take the following (practical) position. We assume that an ontology is represented in a file on the web. Every change that results in a different character representation of the ontology constitutes a revision. In case the logical definitions are not changed, it is the responsibility of the author of the revision to decide whether this revision is semantic change and thus forms an new conceptualization with its own identity, or just an change in the representation of the same conceptualization.

Identification on the web The second question is: how does this relate to web resources and there identity? This is brings us into the very slippery debate on the meaning of URIs, URNs and resources (see Champin et al., 2001, and the discussion on the RDF Interest mailing-list that followed its publication). The main questions in this discussion are: what is a resource and how should it be identified. However, we will circumvent these questions and approach the problem from another direction: are the "entities" in our domain (i.e., the entities in the domain of ontology versions, e.g. a conceptualization, a revision, a specification) resources and can we give them an identifier?

These questions are relatively easy to solve! According to the definition of Uniform Resource Identifiers (URI's) (defined in Berners-Lee et al., 1998), "a resource can be anything that has identity". In (Berners-Lee, 1996) is stated: a "resource" is a conceptual entity (a little like a Platonic ideal). Both definitions comprise our idea of an ontology. Hence, an ontology can harmlessly be regarded as a resource. An URI, which "is a compact string of characters for identifying an abstract or physical resource" (Berners-Lee et al., 1998) can be used to identify the resources. Notice that URI's provide a general identification mechanisms, as opposed to Uniform Resource Locators (URL's), which are bound to the *location* of a resource.

The important step in our proposed method is to separate the identity of ontologies completely from the identity of files on the web that specify the ontology. In other words, the class of ontology resources should be distinguished from the class of file resources. As we have seen above, a revision — which is normally specified in a new file — *may* constitute a new ontology, but this is no automatism. Every revision is a new file resource and gets a new file identifier, but does not automatically get a new ontology identifier.

Notice that we are at this point not compliant with the RDF Schema specification (Brickley and Guha, 2000), which states:[8]

> this specification recommends that a new namespace URI should be declared whenever an RDF schema is changed.

This is recommendation seems too strong, because it also advises to use new URI's when only small corrections are made that do not affect the meaning, i.e., when the conceptualization is not changed. This has already caused problems. For example, the Dublin Core working group changed the URI of their meta-data term definitions when they published a new version with more precisely stated definitions. This has caused a lot of annoyance in the library community, who had to work with several URI's for equivalent concepts. Eventually, the DC steering committee decided to use one URI for all the versions of their definitions.

[8] Although we might be *intentionally* compliant, because the RDFS specification argues that changing the *logical* structure might break depending models. This recommendation could thus be interpreted as only valid for logical changes.

Baseline of an identification method When we take into account all these considerations, we propose an identification method that is based on the following points:

- a distinction between three classes of resources:
  1. files;
  2. ontologies;
  3. lines of backward compatible ontologies.
- a change in a file results in a new file identifier;
- the use of a URL for the file identification;
- only a change in the conceptualization results in a new ontology identifier;
- a new type of URI for ontology identification with a two level numbering scheme:
  - minor numbers for backward compatible modifications (an ontology-URI ending with a minor number identifies a specific ontology);
  - major numbers for incompatible changes (an ontology-URI ending with a major number identifies a line of backward compatible ontologies);
- individual concepts or relations, whose identifier only differs in minor number, are assumed to be equivalent;
- ontologies are referred to by an ontology URI with the according major revision number and the *minimal extra commitment*, i.e., the lowest necessary minor revision number.

The ideas behind these points are the following. As already pointed out in the beginning of this section, the distinction between ontology identity and file identity has the advantage that file changes and location changes (e.g., copy of an ontology) can be isolated from ontological changes. By using a new type of URI, it is possible to encode all the information in it that is necessary for our usage, and it also prevents confusion with URL's that specify a location.

The distinction between individual ontologies on the one hand and lines of backward compatible ontologies on the other hand, provides a simple way to indicate a very general type of compatibility, likewise the "BACKWARD-COMPATIBLE-WITH" field in SHOE (Heflin and Hendler, 2000). The distinction we make is also in line with the idea of "levels of generality", which is discussed in (Berners-Lee, 1996). Applications can conclude directly — without formal analyses or deduction steps — that a version can be validly used on data sources with the same major number and a equal or lower minor number. To achieve a maximal backward compatibility, we also propose that not the minor number of the newest revision is specified in a data source, but the minimal addition to the base version that is used by this data source. For example, suppose an ontology with concepts A, B and C. Version 1.1 added a concept D and version 1.2 added concept E. Then a data source data only relies on concepts A, C and D, would specify its commitment only to version 1.1, although there is already a version 1.2 available. We adopted this idea from software-program library versioning, as described in (Brown and Runge, 2000).

An interesting point for discussion is whether it would be possible to specify the *real* ontological commitment, instead of only the necessary extra commitment. This could lead to even more detailed decisions on compatibility. In our example, this would mean that the data sources specifies that it relies on exactly A, C and D. This would require a different type of identification.

The point that states that individual concepts with a identifier that only differs in minor number are considered to be equivalent, is necessary to actually enable the backward compatibility. By default, all resources on the web with a different identifier are considered to different. This statement allows the creation of a stand-alone ontology revision, which has concepts that are equal to a previous version.

5.2 Change specification and transparent evolution

For change specification and transparent evolution, there are two important requirements. First, because of the suggested practice of referring to the minimal extra addition, the changes in a line of backward compatible ontologies should be easily recognizable and identifiable. Actually, the additions from one version to another together form a *class* of descriptions. Our suggestion is to make this explicit by adding a class `Additions<Major>.<Minor>`, e.g., `Additions1.2`, of which the new descriptions are an instance. This makes that class a unique identifier for the set of additions in a certain revision. The additions can be retrieved by asking for all instances of a specific "Addition" class.

Second, the relation to a previous version should explicitly be specified. One aspect of this specification is a pointer to the ontology from which it is derived. These pointers together form a lattice of versions that can be used to deduce the derivation relation from one version to an arbitrary other version of the *ontology*. A second aspect of this specification is the relation between *concepts and relations* in the previous and current version of the ontology. As concepts and relations that do not have the same major number in their identifier are assumed to be different, this specification should both specify equivalence relations as subsumption relations. However, although a lot of relations between revisions of concepts can be specified with "subclass-of" and "equivalence" relations, a more extensive (rule-)language is necessary to enable the efficient specification of the relations, e.g. a language with quantifiers.

We have seen that an explicit specification of the relation between concepts allows to retain as much information as possible. The relation between two revisions of an ontology can be specified in a separate translation ontology. Both the previous version of the ontology and the translation ontology should be linked from the new version, to allow the automatic collection of all the statements that specify the relation.

6 Conclusions and further work

In this paper, we have explored the problem of ontology versioning in a web based context. We discussed the nature of ontology changes and looked at the consequences. Ontology versioning shares some characteristics with database schema versioning and program library versioning, but also as some peculiarities which are specific for the web based context, mainly introduced by the decentralized nature of the web.

After examining the effects on compatibility of a few example scenarios for ontology versioning, we have sketched some elements for a versioning framework for ontologies. This elements are mainly focused on identification and referring. Our main goal in the design of a framework is to achieve "maximal use" of the available knowledge. This implies that is is not sufficient to find out whether a specific interpretation of a ontology on data is invalid, but that we try to derive as much valid information as possible.

The ideas that are presented in this paper will be implemented in the On-To-Knowledge

project (Fensel et al., 2000b), which builds an ontology-based tool environment to perform knowledge management, dealing with large numbers of heterogeneous, distributed, and semi-structured documents typically found in large company intranets and the World-Wide Web.

The work described in this paper is still ongoing. There are a lot of things that are not yet done. We think that the most important flaw is the lack of a detailed analysis of the effect of specific changes on the interpretation of data. This could be done in the same line as (Banerjee et al., 1987), where effects of schema changes on OO databases are analyses. This analyses should also cover the problem of data that becomes inconsistent. Further, the role of time is also not taken into account. Finally, a clear example is needed to illustrate the proposals.

Eventually, this work will result in a versioning framework that gives very detailed procedures to allow evolving ontologies, while achieving maximal compatibility.

References

Banerjee, J., Kim, W., Kim, H.-J., and Korth, H. F. (1987). Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD Record (Proc. Conf. on Management of Data)*, 16(3):311–322.

Berners-Lee, T. (1996). Generic resources. Design Issues.

Berners-Lee, T., Fielding, R., and Masinter, L. (1998). RFC 2396: Uniform Resource Identifiers (URI): Generic syntax. Status: DRAFT STANDARD.

Bray, T., Hollander, D., and Layman, A. (1999). Namespaces in xml. `http://www.w3.org/TR/REC-xml-names/`.

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

Brown, D. J. and Runge, K. (2000). Library interface versioning in solaris and linux. In *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, Georgia.

Champin, P.-A., Euzenat, J., and Mille, A. (2001). Why urls are good uris, and why they are not. `http://www710.univ-lyon1.fr/~champin/urls.pdf`.

Clark, P. and Porter, B. (1997). Building concept representations from reusable components. In *Proceedings of the AAAI'97*, pages 369–376.

Corcho, O. and Gómez-Pérez, A. (2000). A roadmap to ontology specification languages. In Dieng, R. and Corby, O., editors, *Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International Conference EKAW 2000*, number LNCS 1937 in Lecture Notes in Artificial Intelligence, pages 80–96, Juan-les-Pins, France. Springer-Verlag.

Fensel, D. (2001). Ontologies: Dynamic networks of formally represented meaning. Rejected for SWWS.

Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., and Klein, M. (2000a). OIL in a nutshell. In Dieng, R. and Corby, O., editors, *Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International*

Conference EKAW 2000, number LNCS 1937 in Lecture Notes in Artificial Intelligence, pages 1–16, Juan-les-Pins, France. Springer-Verlag.

Fensel, D., van Harmelen, F., Klein, M., Akkermans, H., Broekstra, J., Fluit, C., van der Meer, J., Schnurr, H.-P., Studer, R., Hughes, J., Krohn, U., Davies, J., Engels, R., Bremdal, B., Ygge, F., Lau, T., Novotny, B., Reimer, U., and Horrocks, I. (2000b). On-to-knowledge: Ontology-based tools for knowledge management. In *Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference*, Madrid, Spain.

Foo, N. (1995). Ontology revision. In Ellis, G., Levinson, R., Rich, W., and Sowa, J. F., editors, *Proceedings of the 3rd International Conference on Conceptual Structures (ICCS'95): Applications, Implementation and Theory*, volume 954 of *LNAI*, pages 16–31, Berlin, GER. Springer.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2).

Heflin, J. and Hendler, J. (2000). Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 443–449. AAAI/MIT Press, Menlo Park, CA.

Klein, M. (2001). Combining and relating ontologies: an analysis of problems and solutions. In Gomez-Perez, A., Gruninger, M., Stuckenschmidt, H., and Uschold, M., editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, Seattle, USA.

Oliver, D. E., Shahar, Y., Musen, M. A., and Shortliffe, E. H. (1999). Representation of change in controlled medical terminologies. *Artificial Intelligence in Medicine*, 15(1):53–76.

Pinto, H. S., Gómez-Pérez, A., and Martins, J. P. (1999). Some issues on ontology integration. In *Proceedings of the Workshop on Ontologies and Problem Solving Methods during IJCAI-99*, Stockholm, Sweden.

Roddick, J. F. (1995). A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393.

Roddick, J. F., Craske, N. G., and Richards, T. J. (1994). A taxonomy for schema versioning based on the relational and entity relationship models. *Lecture Notes in Computer Science*, 823:137–??

Sheth, A. P. and Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236. Also published in/as: Bellcore, TM-STS-016302, Jun.1990.

Ventrone, V. and Heiler, S. (1991). Semantic heterogeneity as a result of domain evolution. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(4):16–20.

A Implementation in DAML+OIL

This appendix gives a rough sketch how the ideas about identification and change tracking might be used in RDF Schema or DAML+OIL. In both languages it is a common practice to use the URL (location) of the ontology as its identifier. This URL is then defined as a namespace, which makes the terms in the ontology identifiable. However, it is not required for the namespace mechanism that *URL's* are used. A namespace is defined as follows (Bray et al., 1999):

> [Definition:] An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names.
>
> ...
>
> [Definition:] URI references which identify namespaces are considered identical when they are exactly the same character-for-character. Note that URI references which are not identical in this sense may in fact be functionally equivalent.

This means that actually *URI's* function as identification mechanism.[9] It is therefore easy to adapt this mechanism for our ontology identification mechanism, which uses a separate URI for ontology identity. We should then first design a URI for ontologies.

According to the URI definition, a "generic URI" has the following format: `<scheme>://<authority><path>?<query>`. Champin et al. (2001) pointed out that the use of a URL as identification guarantees that the publisher has the authorization to use that specific part of the URL namespace. This advantage can be retained when the `<authority>` and `<path>` component in the new URI scheme also use the server name and path to the part of the namespace that is owned by the publisher.

Our suggestion would be to use `ontology` as a name for the scheme, constitute the `<authority>` and `<path>` from the server name and path of a URL, and use the major (and probably a minor number) as last two elements of the path. A typical identifier for an ontology would look as follows: `ontology://www.cs.vu.nl/~{}mcaklein/ontology/example/2/1/`, while an line of backward compatible ontologies is identified by `ontology://www.cs.vu.nl/~mcaklein/ontology/example/2/`.

There is still one important open question. Using URL's as ontology identifiers have the advantage that localization of the file that specifies the ontology is trivial. With separate URI scheme, we cannot use the URI of the ontology to locate the file that specifies it. There are several solutions for this problem. One would be to have some kind of lookup system, like a DNS for host names and IP-numbers. A practical solution would be to provide a direct mapping from ontology URI's to file URL's. For example, one could agree that when "`ontology`" is replaced by "`http`" in a ontology URI, one acquire a URL of a directory with specifications

[9] Remember that URL's are a subset of URI's. The URI definition Berners-Lee et al. (1998) says:

> A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

of the ontology (conceptualization), and that appending `newest.rdfs` to the directory URL yields the URL of the most recent specification. Although we now show that there are several implementation possible, the versioning framework should specify exactly how people and applications should behave with this respect.

Our suggestion is to extend the meta-information in a DAML+OIL ontology also with the *location* of the ontology specification. Together with an identifier, a pointer to the previous version and the the "translation" ontology, a piece of an DAML+OIL ontology might look as follows:

```
<Ontology rdf:about="">
  <identifier>ontology://www.cs.vu.nl/~mcaklein/ontology/example/2/1/</identifier>
  <derivation>
    <from rdf:resource="ontology://www.cs.vu.nl/~mcaklein/ontology/example/2/1/" />
    <relation rdf:resource="ontology://www.cs.vu.nl/~mcaklein/ontology/example/2/1/delta/" />
  </derivation>
  <location>http://www.cs.vu.nl/~mcaklein/ontology/example/2/1/base.rdfs</location>
  <info>${Id}: base.rdfs,v 1.15 2001/05/22 10:26:46 mcaklein Exp $</info>
</Ontology>

...

<rdfs:Class rdf:ID="Additions2.1">
</rdfs:Class>

<rdfs:Class rdf:ID="HBO">
  <rdf:type rdf:resource="#Additions2.1"/>
</rdfs:Class>
```