Ontology Evolution and Versioning

The state of the art

Burcu Yildiz

Vienna University of Technology
Institute of Software Technology & Interactive Systems (ISIS)

Asgaard-TR-2006-3 October 2006

Authors: **Burcu Yildiz**

yildiz@asgaard.tuwien.ac.at
http://ieg.ifs.tuwien.ac.at

Contact: **Vienna University of Technology**
Institute of Software Technology & Interactive Systems (ISIS)

Favoritenstrasse 9-11/188
A-1040 Vienna
Austria, Europe

Telephone:	+43 (1) 588 01 - 18801
Telefax:	+43 (1) 588 01 - 18899
Web:	http://ieg.ifs.tuwien.ac.at

Abstract

Ontologies, are explicit specifications of conceptualisations, and as such serve as a backbone of many Information Systems (ISs) as knowledge bearing artefacts representing mainly domain knowledge. As the use of ontologies in several kinds of ISs increased in the recent years significantly, the question of how to maintain these ontologies, gained more importance.

Although, there are no sophisticated methods available yet to support all the aspects of change management for ontologies, it surely is an active research field. Most of the work has been done under the titles of Ontology Evolution and Versioning. Some methods emerged, which address particular aspects related to the change management of ontologies.

This report gives an overview of the state of the art of research done so far in the fields of Ontology Evolution and Versioning.

Contents

1 Introduction

The invention of computers made many things easier, for example to generate, save, and access data. The invention of the internet was 'the' step to share all this generated data with everyone around the world. Despite many benefits, this development also gave rise to the problem of extracting relevant information out of the overwhelming amount of data we are facing on a daily basis.

The Artificial Intelligence (AI) community has been dealing with this problem for some time now. The research field that comprises all the work in this area is called Information Processing. One particular sub-field deals with the extraction of certain types of relevant information from mainly text documents, which is called Information Extraction (IE).

The question of what actually is 'relevant' information we are looking for, comes to ones mind immediately. Not only is this question hard to answer when complicated task specifications and domains are involved, it is also a problem to communicate this answer to a computer.

Ontologies, being explicit specifications of conceptualisations [Gruber, 1993], can be used in that context to provide Information Extraction Systems (IES) with the definition of relevant information in a formal and computer-understandable way. Ontologies gained more attention lately, as many researchers think that they can serve as the backbone of the Semantic Web. The Semantic Web is an extended form of the current Web with semantics attached to data, which can make it easier for humans and machines to find information [Berners-Lee, 1999]. But the application areas of ontologies are not bound to the use in the Semantic Web. They are knowledge bearing artifacts and can be used in any application area where a domain of interest has to be conceptualised.

However, the use of ontologies in Information Systems (IS) requires an accurate management of the same. The most important issue in that context is to keep the ontology up-ot-date w.r.t. the changes in its domain of usage, since most domains have a dynamic nature.

In this work, we will explore the state of the art of research in the field of change management for ontologies. For that purpose, we have to state first what an ontology actually is and why it is important for many of todays ISs.

2 Ontologies

Originated in early Greece, the term 'Ontology' is a branch of philosophy that deals with the nature and organisation of being. Philosophers like Plato and Aristotle dealt with this branch, trying to find the fundamental categories of being and to determine whether the items in those categories can said to 'be'. The proper naming of the items was questioned by Plato. In his opinion, the item names in an optimal world, would refer in everybody's minds to one and only one thing.

The computer scientists community is, in general, not that much interested in philosophy, but the idea a means for representing fundamental categories of a particular domain to establish a common understanding was worth considering. So it happened that ontologies became very popular in Artificial Intelligence and Knowledge Representation. Here, an ontology has been seen as yet another "*engineering artefact*, constituted by a specific *vocabulary* used to describe a certain reality, plus a set of explicit assumptions regarding the *intended meaning* of the vocabulary words" [Guarino, 1998].

In this section, I will first define the basic terms with regard to ontologies and will give a short insight into possible benefits ontologies can yield to. Further, I will explain the role of formal logics in representing ontological knowledge and will give an overview of the most common ontology representation languages.

2.1 Definition

The literature contains many, partly contradicting, definitions of an ontology. [1] However, the best-known and most quoted definition in the AI community became the definition by Gruber [1993], which is also the one I commit to in this thesis.

An ontology is an explicit specification of a conceptualization.

The term *conceptualisation* refers to an abstract model, a simplified view of a particular domain of concern. By defining the concepts, relations, and the constraints on their use in a formal way, the conceptualisation becomes explicit.

The definition by Gruber has been thoroughly analysed by Guarino and Giaretta [1995]. First of all, they stated that it is crucial to distinguish between the 'Ontology' with a capital 'O' as a branch of philosophy which deals with the nature and the organisation of being; and the 'ontology' as a particular object (engineering artefact). They, further specified seven possible interpretations of the term ontology and elucidate the implications of such various interpretations:

- Ontology as a philosophical discipline

[1]Note that ontologies were introduced to get a clear and common view on things, and yet the community cannot agree on the definition of the term itself.

- Ontology as an informal conceptual system
- Ontology as a formal semantic account
- Ontology as a specification of a "conceptualization"
- Ontology as a representation of a conceptual system via a logical theory characterized by specific formal properties or only by its specific purposes
- Ontology as the vocabulary used by a logical theory
- Ontology as a (meta-level) specification of a logical theory

Interested readers are also referred to the article of Zúñiga [2001], who explained, in pretty much detail, the differences between the philosophical meaning of ontologies and the meaning in Information Systems by giving a deep analysis and comparison on both, Gruber's and Guarino's views.

We will use the following formal definition of an ontology in the rest of this thesis, whenever referring to ontological components.

Definition 1 *An ontology is a tuple* $\mathcal{O} := \{\mathcal{C}, \mathcal{R}, \mathcal{H}^{\mathcal{C}}, \mathcal{H}^{\mathcal{R}}, \mathcal{I}_{\mathcal{C}}, \mathcal{I}_{\mathcal{R}}, \mathcal{A}^{\mathcal{O}}\}$*, whereas*

• $\boldsymbol{\mathcal{C}}$***:*** *represents a set of concepts.*

• $\boldsymbol{\mathcal{R}}$***:*** *represents a set of relations that relate concepts to one another.* $R_i \in \mathcal{R}$ *and* $R_i \rightarrow \mathcal{C} \times \mathcal{C}$.

• $\boldsymbol{\mathcal{H}^{\mathcal{C}}}$***:*** *represents a concept hierarchy in form of a relation* $\mathcal{H}^{\mathcal{C}} \subseteq \mathcal{C} \times \mathcal{C}$*, whereas* $\mathcal{H}^{\mathcal{C}}(C_1, C_2)$ *means that* C_1 *is a subconcept of* C_2.

• $\boldsymbol{\mathcal{H}^{\mathcal{R}}}$***:*** *represents a relation hierarchy in form of a relation* $\mathcal{H}^{\mathcal{R}} \subseteq \mathcal{R} \times \mathcal{R}$*, whereas* $\mathcal{H}^{\mathcal{R}}(R_1, R_2)$ *means that* R_1 *is a subrelation of* R_2.

• $\boldsymbol{\mathcal{I}_{\mathcal{C}}}$ ***and*** $\boldsymbol{\mathcal{I}_{\mathcal{R}}}$***:*** *represent two disjoint sets of concept and relation instances, respectively.*

• $\boldsymbol{\mathcal{A}^{\mathcal{O}}}$***:*** *represents a set of axioms.*

Note, that an ontology can be considered in two parts: an extensional and an intensional part. The extensional part is represented by $\mathcal{C}$, $\mathcal{R}$, $\mathcal{H}^{\mathcal{C}}$, $\mathcal{H}^{\mathcal{R}}$, and the intensional part is represented by $\mathcal{I}_{\mathcal{C}}$ and $\mathcal{I}_{\mathcal{C}}$.

2.2 Characteristics of Ontologies

In general, ontologies consist of a set of concepts and a description of the relationships that hold between these concepts. But when examining them closer one can see many differences between them. Many researchers have described characteristics of ontologies in order to classify them. The names of these characteristics vary from researcher to researcher, but they can be grouped as follows:

2.2.1 According to their level of formality

The same conceptualisation can be defined by using different languages. This yield ontologies with different levels of formality because the underlying languages have different levels of formality. Uschold and Grueninger [1996] classified ontologies in four groups according to this characteristic:

- *Highly informal:* if they are expressed in natural language.
- *Semi informal:* if they are expressed in restricted and structured natural language.
- *Semi formal:* If they are expressed in a semi-formal defined language.
- *Rigourously formal:* If they are expressed in a rigourously formal defined language.

If the ontologies are going to be used to share knowledge between humans than it is better to have an informal or highly informal ontology. But if the interaction partners are going to be computers it would be better to have at least a semi-formal or even a rigorously formal ontology.

2.2.2 According to their level of generality

An ontology can contain information with different levels of detail. The distinction made by Guarino [1998] reflects this thought:

- *Reference (off-line) ontologies:* These are detailed ontologies, which get closer to specifying the intended meaning of a vocabulary. They are used only from time to time for referencing.
- *Coarse (on-line) ontologies:* They consist of a minimal set of axioms intended to be shared by different sites.

Gruber [1998] have made a similar distinction:

- *Top-level ontology:* Such ontologies describe very general concepts, which are independent of a particular domain or task like as space, time, event, action, etc.

- *Domain ontology:* Such ontologies contain a description of a vocabulary to a generic domain like as medicine or automobiles, etc. They are a specialisation of the terms in the top-level ontology.

- *Task ontology:* Such ontologies contain a description of vocabulary to a generic task or activity like as diagnosing or selling, etc. They are a specialisation of the terms in the top-level ontology.

- *Application ontology:* Such ontologies are bound to both, a specific domain and task, and are often a specialisation of both the related ontologies.

2.3 Why to bother about Ontologies?

Ontologies are means for an agreement on the meaning of 'things' between interaction partners, either humans or computers. By committing to an ontology, an interaction partner declares that he is aware of the meaning of the vocabulary words in the ontology and that he will share this meaning during their interaction, which guarantees consistency during the interaction. To commit to an ontology should not require any changes in the working environments of the interaction partners, though.

Because ontologies can be used to share a specific conceptualisation among communication partners, its usage is theoretically beneficial wherever an agreement on the meaning of things is important.

First of all, an ontology provides interaction partners with a common vocabulary and standardises in this way the used terminology. Further, the knowledge that each interaction partner usually use implicitly, can be made explicit with an ontology which makes it possible to share and reuse this knowledge. Based on these thoughts, we can list some benefits of ontology usage as follows:

- *Understanding:* An ontology can serve as a documentation with which human beings can understand an underlying conceptualisation better.

- *Communication:* Ontologies can help interaction partners to communicate over a domain of interest in an unambiguous way. For this purpose, interaction partners could either send their respective ontologies to one another or commit to a shared ontology.

- *Inter-operability:* Computer systems can inter-operate in a consistent manner. This could enable interaction partners to inter-operate across organisational and/or international boundaries.
- *Reuse:* There is no need to reinvent the wheel over and over again. Therefore, ontologies as knowledge components should be reused by others. Menzies [1999] stated that the reuse of ontologies can be compared to the reuse of already programmed modules in software engineering. In many cases it took so much time to understand and integrate the module that 60 percent of it could have been written from scratch. Whether this applies to ontologies as well, is yet to be empirically analysed. Menzies statement could be true for relatively small ontologies, but in the case of very large ontologies that have been developed over years (e.g., WordNet, MeSH) it could be hard to write them from scratch.

Some researchers are taking the use of ontologies critically, and doubt that ontologies will bring significant benefits as promised. A cost-benefit analysis should be done to clarify this issue. However, there is no empirical study available yet, because ontologies are not in use for so long (for example, in a long-term project).

Despite such sceptical views, ontologies are in use in many kinds of ISs for different purposes by now. According to Guarino [1998] their use can be beneficial during the development time and run time of an IS. The usage during development time enables the developer to practise a higher level of reuse. For the usage at run time he distinguishes between to types: ontology aware IS and ontology driven IS. Ontology aware ISs are systems where the system was build by keeping the ontology in 'mind' and where the ontology is at hand for usage. Ontology driven ISs, at the other hand, include the ontology as yet another component of the system which contributes to the overall IS goal.

However, the use of ontologies in ISs, no matter in which particular form, requires an accurate management of the same. The most important issue in that context is to keep the ontology up-ot-date w.r.t. the changes in its domain of usage, because most domains are dynamic by nature. In the rest of this report, we will thus examine the requirements to an accurate change management for ontologies in more detail.

3 Change Management for Ontologies

As mentioned earlier (see Section 2.1), ontologies are abstract views of specific fields of concern. These abstract views cannot be considered as static, because there are several occasions that can make it necessary to change the ontology. Such occasions may be corrections in the conceptualisation, adapting the ontology to changing facts in the real world in order to catch up with the current reality, etc.

Klein and Noy [2003] define several tasks that have to be provided for a complete change management support, as follows:

- *Data Transformation:* A change in the extensional part of an ontology (i.e., $\mathcal{C}$, $\mathcal{R}$, $\mathcal{H}^{\mathcal{C}}$, $\mathcal{H}^{\mathcal{R}}$ in Definition 2.1) may require other changes in the intensional part of the ontology (i.e., $\mathcal{I}_{\mathcal{C}}, \mathcal{I}_{\mathcal{R}}$ in Definition 2.1). For example if two concepts A and B are merged, then the instances of these classes have to be merged as well.
- *Data Access (Compatibility):* It should be possible to access data confirming to the old version via the changed new version as well. This brings the question of 'what kind of data we want to preserve?' to the forefront, because there are several dimensions to compatibility between ontology versions [Noy and Klein, 2004]:
  - Instance-data preservation: to make sure that no data is lost during the transformation from the old version to the new one.
  - Ontology preservation: to make sure that a query result using the new version is a subset of the same query result using the old version.
  - Consequence preservation: to make sure that all the facts that could be inferred from the old version can still be inferred from the new version.
- *Ontology Update:* In a distributed environment, where an ontology is distributed to many users, it should be possible for users to update their local ontologies when the corresponding remote ontology changes.
- *Consistent Reasoning:* The consistency of an ontology should be maintained after changes occur. This will ensure that reasoning is still possible on the changed ontology.
- *Verification and Approval:* Interfaces to enable and simplify the validation and verification of proposed changes to an ontology, by ontology developers should be provided.

Change management in ontologies can take several forms:

- **Ontology Modification:** changing the ontology without bothering about its consistency.
- **Ontology Versioning:** building and managing different versions of an ontology and providing access to these versions.
- **Ontology Evolution:** changing the ontology while keeping it consistent.

Obviously, the terms 'ontology versioning' and 'ontology evolution' have been adopted by the ontology engineering community, as many researchers thought that these fields are similar to the fields of schema evolution and versioning in the database community. Although, many other researchers stated that there are fundamental differences between those two fields and it is not possible to strictly distinguish between ontology versioning and evolution [Noy and Klein, 2004], the terms remained and are being used widely still.

In this section, we will introduce the research fields of ontology versioning and evolution in more detail.

3.1 Change Operations

To define change operations for ontologies is not easy, because one has to take into account all the possible effects a change can have on the components of an ontology. According to Klein [2004] one can distinguish between three kinds of changes:

- *Conceptual changes:* represent changes in the conceptualisation itself, for example changing concepts relations, etc.
- *Specification changes:* represent changes with regard to the specifications of the conceptualisations, for example to add new properties to a concept.
- *Representation changes:* represent changes in the representation of the conceptualisation, for example by using another ontology representation language.

Noy and Klein [2004] define also a set of change operations for ontologies considering the effects on the compatibility between two versions of an ontology (see Section 3). Table 1 contains those change operations, when considering an ontology through its traditional elements: classes, slots, slot restrictions, and instances. ('+' means that no data is lost, '-' means that it cannot be guaranteed that no instance data is lost, and '$\sim$' means that no instance data is lost if a part of the data is translated into a new form.)

Table 1: Ontology change operations and their effects on instance data

Operation	**Effect**	**Comment**
Create class C	+	No data is lost.
Delete class C	-	Instances of class C have a less specific type (they have become instances of the superclass of C).
Create slot S	+	No data is lost.
Delete a slot S	-	The values of slot S for all instances. are lost
Attach slot S to class C	+	No data is lost.
Remove slot S from class C	-	The values of slot S for instances of C are lost.
Add a subclass-superclass link between a subclass SubC and a superclass SuperC	+	SubC has new slots inherited from SuperC - in most cases equivalent to adding slots.
Remove a subclass-superclass link between a subclass SubC and a superclass SuperC	-	SubC no longer has the slots inherited from SuperC. The values for these slots for instances of SubC are lost.
Re-classify an instance I as a class	+	No data lost.
Re-classify a class C as an instance	-	Instances of C are less specifically typed.
Declare classes C_1 and C_2 as disjoint	-	Instances that belonged to both C_1 and C_2 are invalid.
Define a slot S as transitive or symmetric	-	Slot values for S that violated the transitivity or symmetry property are invalid.
Move a slot from a subclass SubC to a superclass SuperC	+	Class SubC still inherits slot S.
Move a slot S from a superclass SuperC to a subclass SubC	-	Class SuperC no longer has slot S. The values for slot S for instance of SuperC are lost.
Move a slot S from a class C_1 to a referenced class C_2	$\sim$	No data is lost if the values of the slot are moved.
Encapsulate a set of slots into a new class	$\sim$	No data is lost if the values of the slot are moved.

Table 1 continued		
Operation	**Effect**	**Comment**
Change a superclass of a class C to a class higher in the hierarchy	-	C no longer has the slots it inherited from its direct superclass. The values for these slots for instances of C are lost.
Change a superclass of a class C to a class lower in the hierarchy	+	C has possibly inherited additional slots. No data is lost.
Widen a restriction for a slot S	+	All the existing slot values are still valid.
Narrow a restriction for a slot S	-	Slot values that violated the narrower restrictions are invalid.
Merge classes	∼	No data is lost if values of slots are moved.
Split classes in several classes	∼	No data is lost if values of slots are moved.

3.2 Representing Ontology Changes

One issue in the context of change management is the proper representation of changes. The easiest and the most straight-forward way might be to keep track of changes in form of a change log that contains the exact sequence of changes applied to an ontology. Although easy, it might be a problem to make such change logs available to a distributed community of ontology developers and/or users. However, it can be useful for local ontology development, hence the change log support by several ontology-editing tools like as Protégé or OntoEdit.

According to Klein [2004] a comprehensive change specification should consist of at least the following information:

- an operational specification of a change,
- the conceptual relation between the old and new versions of the changed constructs,
- meta-data about the change,
- the evolution relation between constructs in the old and new version,
- and information about task or domain specific consequences of changes.

Besides change logs, Klein and Noy [2003] mention three more possibilities to access and represent changes between two versions of ontologies: performing a structural diff, representing differences in form of conceptual changes or in form of a transformation set. The first way, is to perform a *structural diff* as described in the work of Noy and Musen [2002]. They draw a comparison to the field of software code versioning, where the code also changes often and differences between two versions can be accessed using a process called *'diff'*, which returns a list of lines that are different in the two versions. The authors state, however, that this approach cannot be inherited for comparing two ontology versions, because the form of representation and the form of generating ontologies is completely different then with software code. So, it is possible that two ontologies are identical in terms of their conceptualisations, but differ immensely in terms of their internal representation. Therefore, they propose an algorithm called PROMPTDIFF[2] for comparing two ontologies w.r.t. their structure and not their text representation. For each ontological structure in the old version of an ontology, it looks for possible corresponding structures in the new version. If there are some structures for which no direct counterparts can be found, several heuristics are being applied to search for possible matches. It then tries to merge these results using a fixed-point algorithm. The authors claim that PROMPTDIFF can achieve an average recall value of 96% and an average precision value of 93%.

The second way, is to represent differences between ontologies in form of *conceptual changes*. Such changes specify the conceptual relation between ontological structures in the old version and the new version. For example, a conceptual change may state that a concept A was a subconcept of B in the old version before being moved to its place in the new version.

The third way, is to represent differences between ontologies in form of *transformation sets*, which contain change operations that specify how (i.e. applying which changes) the old version of an ontology can be transformed into the new version. Transformation sets differ from simple change logs, as they only contain necessary operations to achieve the intended (changed) version and not every single change applied to the ontology as in simple change logs. Furthermore, transformation sets may not contain changes in the same order as they were really applied. For example, adding new components can be grouped together, because they do not affect the rest of the ontology like as delete operations.

We think, that ontology changes can also be integrated into the ontology itself as instances of a general concept 'change'. These instances, then, can be used to save information about affected ontological components. One may think of many other ways to represent ontological changes. The important thing is to choose a way that serves the purpose of its' application most.

[2]This algorithm is available as a plugin for the Protégé 2000 ontology-editing environment.

3.3 Ontology Versioning

Klein [2002] defines Ontology Versioning as "the ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology".

Ontology Versioning should enable the management of different versions of the same ontology at the same time. This functionality is essential in scenarios where developers or users of an ontology are going to access an ontology in a distributed manner. Considering that one of the major benefits of ontologies is the re-use and inter-operabilty they can provide, such a scenario is more than a theoretical one. Currently, no sophisticated versioning mechanisms are available. Often, ontologies change and the old versions are lost forever, because only the latest version is accessible. Sometimes, old and new versions of ontologies are archieved, but no mechanisms are provided to highlight the differences between versions.

The first attempt to address this problem has been taken by Heflin and Hendler [2000] with introducing the Simple HTML Ontology Extensions (SHOE) as an extension to HTML to represent ontology-based knowledge using additional tags. One important facility is that SHOE enables ontology developers to state whether a version is backward-compatible with an old version or not. In a distributed application field, where many interaction partners (e.g., applications, agents, etc.) use the same ontology, this information is essential, because it determines whether they can continue with their work as usual or they have to update their versions in order to maintain agreed-upon interaction. The work of Heflin and Hendler [2000] is also important in terms of its long-standing contribution, by starting the discussion about the problem of ontology versioning in dynamic, distributed, and heterogeneous environments.

However, the current trend of the Semantic Web makes more sophisticated approaches to ontology versioning necessary. Klein [2002] impose the following requirements on an ontology versioning framework:

- *Identification:* The intended definitions of ontological components have to be made clear in advance, because they represent prerequisites for change specifications.

- *Change specification:* Possible changes have to be specified according to the identification of ontological components. Because, different representation paradigms provide different components, the change specifications will also differ.

- *Transparent evolution:* It should be clear what the actions are that have to be taken when particular changes occur. For that purpose, change specifications will be used to translate and relate different versions of ontological components.

- *Task awareness:* Because, there are different dimensions to compatibility (e.g., preservation of instance data, preservation of query results, preservation of consequence, etc.), a framework have to be aware of the concrete task in order to provide appropriate transformations between versions.

- *Support for untraced changes:* It is often the case, that there is no track of changes that represent the steps from one version to the new one. In such cases, a versioning framework should provide mechanisms to determine whether two versions are compatible.

The main objectives of a versioning framework that reconcile the above mentioned requirements can also be found in Klein's work [2002]. The first objective is surely to provide *data accessibility* through different versions of an ontology. This can be achieved, either by restricting allowed changes to only those that do not affect the interpretation of data, or by providing translations between the versions so that user queries can be translated back in order to access the data in the old form.

Consistent reasoning is another objective, as it aims to ensure that reasoning over the ontologies is not affected. In that way, it can be guaranteed that the answers to at least a specific set of queries will remain the same with different versions.

In a distributed environment, it is also important to provide *synchronisation* and *data translation* support. Whereas, the former enables the update of local ontologies with a remotely changed ontology, the latter enables the automatic translation of affected data sources to conform to a newer version of an ontology.

Versioning is also important w.r.t. collaborative ontology development, where more than one developer wants to make changes on an ontology (*management of development*). For such a scenario, step-by-step verification and authorisation have to be supported.

3.4 Ontology Evolution

Stojanovic and colleagues [2002] define Ontology Evolution as follows:

> Ontology Evolution is the timely adaptation of an ontology to changed business requirements, to trends in ontology instances and patterns of usage of the ontology-based application, as well as the consistent management/propagation of these changes to dependent elements.

They further [2002] formulated a set of design requirements for proper ontology evolution:

- It has to enable resolving the given ontology changes and to ensure the consistency of the underlying ontology and all dependent artefacts;
- It should be supervised allowing the user to manage changes more easily;
- It should offer the user advice for continual ontology refinement.

According to them the ontology evolution process can be considered in six phases (see Figure 1):

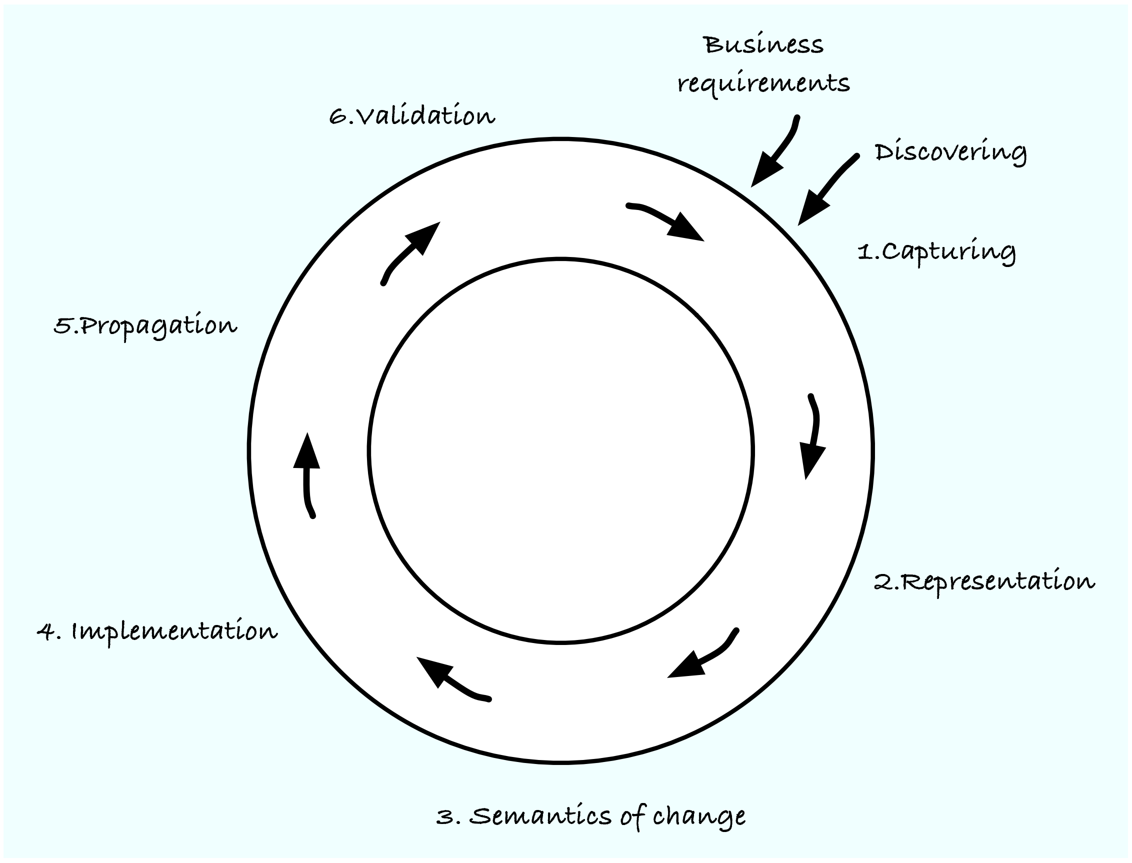

Figure 1: Phases of Ontology Evolution

- *Change capturing :* This phase encapsulates the process of deciding to apply a change on an ontology. This might be forced by explicit requirements (the ontology engineer decides to make a change) or by results of automatic change discovery methods. The first kind of changes are called top-down changes, whereas the second one are called bottom-up changes. Bottom-up changes can be proposed by three different approaches to change discovery: structure-driven, data-driven, and usage-driven change discovery [Stojanovic and Motik, 2002].

- *Change representation:* In order to resolve changes, they should be identified and represented clearly and in a suitable format. They can be represented as elementary or complex changes.

- *Semantic of changes:* How a change can affect the ontology's consistency must be understood in advance, whereas the meaning of consistency depends on the underlying ontology model.

- *Change propagation:* To preserve consistency, affected artefacts should be handled appropriately as well. In a distributed environment, affected artefacts are not bound to local components of the changed ontology, but contain also distributed ontologies that reuse or extend the changed ontology, or even applications that are based on the changed ontology.

- *Change implementation:* Before applying a change, all implications of it have to be presented to the user, who then can accept or discard it. If the user agrees with the changes, all activities to apply the change have to performed.

- *Change validation:* It should be possible for a user to validate performed changes and to reverse the effects of them when necessary.

It is essential to discover the types of changes that can occur, because they have to be handled differently. We distinguish between basic (or elementary) changes, like as deleting or adding a concept, and complex (or composite) changes that are composed of multiple basic change operations.

More important is the distinction between changes that can lead the ontology into an inconsistent state and changes that cannot. Whereas the former class of changes do not cause any problems, the latter class of changes require special treatment. Such a treatment can be in form of an evolution strategy, set by the user in advance to define how to resolve critical changes unambiguously. Stojanovic and Motik [2002] stated the situations in which an **evolution strategy** could help to determine the further course of action:

- how to handle orphaned concepts

- how to handle orphaned properties

- how to propagate properties to the concept whose parent changes

- what constitutes a valid domain of a property

- what constitutes a valid range of a property
- whether a domain of a property can contain a concept that is at the same time a subconcept of some other domain concept
- the allowed shape of the concept hierarchy
- the allowed shape of the property hierarchy
- must instances be consistent with the ontology

Stojanovic and colleagues [2002] introduced also the term of **advanced evolution strategy**. It represents a mechanism that automatically combines available elementary evolution strategies and relieves users of choosing them individually. They defined the following set of advanced evolution strategies:

- *Structure-driven strategy:* resolves changes according to criteria based on the structure of the resulting ontology.
- *Process-driven strategy:* resolves changes according to process of changes itself.
- *Instance-driven strategy:* resolves changes to achieve an explicitly given state of the instances.
- *Frequency-driven strategy:* applies the most used or last recently used strategy.

4 Tool Support for Ontology Change Management

The life-cycle of an ontology consists of several phases that represent difficult and dynamic activities. Starting with the design and generation of an ontology, over to accurate change management to keep the ontology up-to-date and at the same time consistent, till to the evaluation of the ontology; each phase has emerged as a research field of its own.

Several tools have been generated in order to support ontology developers while performing these activities. However, most of them provide support for only restricted aspects of the whole engineering process, mostly the generation phase. But, we have seen that ontologies are rather dynamic artefacts, so there is a need to provide also support for change management of ontologies. Stojanovic and Motik [2002] stated the following groups of requirements that tools for change management have to fulfil:

- *Functional requirements:* Ontology change operations that are going to be supported by the tool have to specified in advance. This set of change operations depend on the underlying ontology model, because different ontology models provide different modeling primitives. For each modeling primitive their should be change operations for adding new ones, deleting and modifying existing ones.

- *User's supervision requirement:* Users should have the right to resolve changes by themselves. Therefore, appropriate mechanisms are required to resolve changes in a way that they do not affect the consistency of an ontology. One such mean are evolution strategies, which users can set up in advance to specify the course of action when 'critical' changes are going to be applied and which can yield to an inconsistent ontology.

- *Transparency requirement:* In order to provide transparency, users should be informed about the effects of a change before the change is going to be applied.

- *Reversibility requirement:* It should be possible to reverse the effects of a change. This, in turn, requires the specification of how particular ontology change operations can be undone.

- *Auditing requirement:* The changes applied to an ontology should be recorded in some way to preserve the history of changes. Such a history will be mainly used to provide reversibility.

- *Ontology refinement requirement:* Continual refinement of an ontology should be provided. This includes the semi-automatic detection of potential changes that could improve the ontology, either by analysing ontology-based data (structure-driven and data-driven approach) or the user's behaviour (usage-driven approach).

- *Usability requirement:* Users should be supported during change management by providing them with information about ontology inconsistencies and about ways to solve them. Further, appropriate visualisations regarding important change management issues (e.g., inconsistency alert, implications of a change, etc.) could help them to discover and manage problematic situations better.

In this section, we will explore to what extent some prominent ontology-editing tools provide change management support for ontologies. Further, we will take a look at stand-alone tools that aim to provide change management support for ontologies.

4.1 KAON Framework

The KArlsruhe ONtology Management Infrastructure (KAON) is an open-source tool that was developed by the members of the Institute AIFB at University Karlsruhe and the Research Center for Information Technology (FZI). It can be downloaded from the webpage of the Karlsruhe Ontology and Semantic Web Tool Suite[3]. The KAON tool itself provides functionalities such as creation, storage, retrieval, and maintenance of ontologies and it is based on RDF(S). There are also some modules which extend the functionalities of KAON, namely KAON Extensions .

KAON is based on several modules that make the creation and management of ontologies possible. There is for instance the front-end module which consists of two user-level applications: the KAON Workbench and the KAON Portal. The KAON Workbench provides a user interface for several ontology-based applications, whereas the KAON Portal is an application for ontology-based web portals.

The KAON Workbench includes the OI-Modeler and the Open Registry. The OI-Modeler is an ontology editor using which one can load an existing OI-model, create a new one, or copy an existing model to a new one. After making a change to the ontology the OI-Modeler lists a set of changes which would be actually necessary to achieve the desired result while keeping the ontology consistent. To support undo and redo functionality, KAON generates evolution logs for all applied changes. Whereas changes like adding a concept to the ontology may not lead to inconsistencies in the ontology, other changes can do (e.g., deleting a concept). To

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

deal with such situations, KAON supports the specification of evolution strategies. The following list contains possible different situations and courses of action predefined by an evolution strategy:

- Orphaned concepts will be: deleted, reconnected to ontology root, reconnected to superconcepts.
- Orphaned properties should be: deleted, reconnected to superproperties, left as they are.
- When concept's parent is removed: properties will not be propagated, all inherited properties will be added to the concept, only parent's properties will be added to the concept.
- Domain/range of a property: may contain subconcepts of other domain/range concepts, may not contain subconcepts of other domain/range concepts.
- Properties without any domain concepts: may exist in the OI-model, should be deleted from the OI-model.
- Properties without any range concepts: may exist in the OI-model, should be deleted from the OI-model.
- Instance consistency: should be enforced, should not be enforced.
- When creating a hierarchy path which already exists: nothing special should be done, the shorter path should be removed, an error should be raised.
- When concept is removed: instances should be removed, instances should be reconnected to the superconcepts, only unique instances should be deleted.
- When property is removed: property instances should be removed, property instances should be defined for the parent properties.

KAON can be considered as one of the ontology-editing environments that provide relatively good change management support for ontologies, by providing full undo/redo functionality, evolution strategies, a consistency checker, and transparency of its actions.

4.2 OilEd

OilEd is a free ontology editor developed at the University of Manchester. It supports DAML+OIL, but its latest version also supports OWL. The tool can be downloaded from the project website[4].

OilEd cannot be seen as an enhanced ontology management system, because it does neither support the development of large-scale ontologies, nor the migration, integration, or versioning of ontologies. Rather it provides a simple editing environment with enough functionality to build ontologies and keep them consistent by using the FaCT reasoner [**?**]. Although it keeps a log-file about all activities with the tool, it does not provide undo/redo functionality.

4.3 Protégé

The Protégé system was developed by the interdisciplinary academic and research group Stanford Medical Informatics (SMI) within the Department of Medicine in the Stanford University School of Medicine. Its history goes back to the first Protégé meta-tool for knowledge-based systems built in 1987. Since the first release of Protégé, namely Opal, there were further three releases (Protégé-I, Protégé-II, and Protégé/Win) until the current system Protégé-2000 was released, which can be freely downloaded on the project website[5]. Protégé-2000Õs knowledge model is based on the Open Knowledge Base Connectivity (OKBC) model, which provides a uniform model of Knowledge Representation Systems (KRS) and is more flexible than other knowledge models used by earlier releases of Protégé. It stores its knowledge in a special-purpose flat file format. It allows you to read from and write to PDF files and a relational database format.

Protégé-2000 allows users to make changes on an ontology and provides undo/ redo functionality. Furthermore, it is possible to archive different versions of an ontology and to revert to previous versions. It is one of the few tools that support ontology versioning, whereas this functionality can be further enhanced by integrating the PROMPTDIFF plugin developed by Noy and Musen [2002]. Using this plugin, users can see the structural differences between two versions of an ontology.

4.4 OntoView

OntoView is a web-based system that supports versioning of online ontologies [Klein *et al.*, 2002]. OntoView enables users to keep interoperable versions of an

[4]http://oiled.man.ac.uk/

[5]http://protege.stanford.edu/

ontology, by highlighting differences between versions and allowing the user to specify the conceptual relation between them.

To provide a transparent interface to arbitrary versions of ontologies, OntoView keeps track of the conceptual relations and transformations between components of the ontology among different versions. This additional information, which is stored explicitly, can be exported as a separate mapping ontology, which in turn can be used as addapters for data sources or other ontologies that are based on the changed ontology.

One of its main functions is *identification of ontologies.* As ontologies are explicit specifications of conceptualisations shared by interaction partners, it is required to have a unique and stable identifier, which can be used by humans or agents to refer to an ontology in an unambiguous way. OntoView supports the identification of ontologies through namespaces.

Further, OntoView provides basic support for the *analysis of possible effects of changes*, by highlighting affected critical components in the ontology.

5 Conclusions

You may have noted that the references section contain mostly work from a handful of people. This is an indicator that research in the fields of Ontology Evolution and Versioning is still in its early stages.

One of the first things that have to be cleared in both research fields are the concrete tasks they cover. Available definitions for Ontology Evolution and Versioning describe overlapping objectives, which, on the other hand, could yield to the interpretation that there is no strict distinction between them.

The basic component on which both research fields concentrate, are ontology change operations. However, it is not easy to state the definitions of ontology changes, because they are highly determined by the representation paradigm they are based on. So an analysis of change definitions for different ontology representation languages have to be made.

Further, it is important how the change operations will be represented themselves. Current techniques are ranged from simple log-files, to conceptual changes, over to transformation sets that specify what to do in order to get the new version starting with the old version.

As the little survey on ontology editing tools showed (see Section 4), the support of change management for ontologies is also scarce. Tools generally provide support for only a few aspects of ontology change management. Most of them provide undo/redo functionality by keeping log-files about the users' activities. Some of them also provide consistency checking to aid ontology developers during generation and editing. However, tools are needed, which are able to maintain ontologies with respect to all change management related aspects.

It is also desirable to provide automatic change management. Automatic change management would require first of all automatic change detection and propagation mechanisms.

Research in these fields can contribute to a wider acceptance of ontologies in Information Systems (ISs), as they would enable better management of ontologies, what is essential for many ISs, because their application fields are in general dynamic by nature and it is not realistic to assume that an ontology will not undergo changes over time.

References

[Berners-Lee, 1999] T. Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper San Francisco, 1999.

[Gruber, 1993] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43:907–928, 1993.

[Guarino and Giaretta, 1995] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In Mars N, editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32. IOS Press, Amsterdam, The Netherlands, 1995.

[Guarino, 1998] N. Guarino. Formal ontology and information systems. In Nicola Guarino, editor, *Proceedings of the First International Conference on Formal Ontologies in Information Systems (FOIS)*, pages 3–15, June 1998.

[Heflin and Hendler, 2000] J. Heflin and J.A. Hendler. Dynamic ontologies on the web. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000.

[Klein and Noy, 2003] M. Klein and N. Noy. A component-based framework for ontology evolution. Technical report, Department of Computer Science, Vrije Universiteit Amsterdam, 2003.

[Klein *et al.*, 2002] M. Klein, A. Kiryakov, D. Ognyanoff, and D. Fensel. Finding and specifying relations between ontology versions. In *Proceedings of the ECAI-02 Workshop on Ontologies and Semantic Interoperability*, 2002.

[Klein, 2002] M. Klein. Versioning of distributed ontologies. Technical report, Vrije Universiteit Amsterdam, 2002.

[Klein, 2004] M. Klein. *Change Management for Distributed Ontologies*. PhD thesis, Department of Computer Science, Vrije Universiteit Amsterdam, 2004.

[Menzies, 1999] T. Menzies. Cost benefits of ontologies. *Intelligence*, 10:26–32, 1999.

[niga, 2001] G.L. Zú niga. Ontology: Its transformation from philosophy to information systems. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, pages 187–197, October 2001.

[Noy and Klein, 2004] N. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6(4):428–440, 2004.

[Noy and Musen, 2002] N. Noy and M. Musen. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *Proceedings of the National Conference on Artificial Intelligence*, 2002.

[Stojanovic and Motik, 2002] L. Stojanovic and B. Motik. Ontology evolution within ontology editors. In *Proceedings of the OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management*, pages 53–62, September 2002.

[Stojanovic *et al.*, 2002] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 285–300, October 2002.

[Uschold and Grüninger, 1996] M. Uschold and M. Grüninger. Ontologies: Principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.