An Ontology Versioning Framework

Tomi Kauppinen

Helsinki 14th June 2004

Master of Science Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Tomi Kauppinen

Työn nimi — Arbetets titel — Title

An Ontology Versioning Framework

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master of Science Thesis	14th June 2004	50 pages

Tiivistelmä — Referat — Abstract

Knowledge is produced and represented increasingly in some "understandable" machine format, where different pieces of the knowledge are related with other pieces of knowledge. Ontologies aim to provide means for describing concepts of the world and the relations between them. An ontology combines classes, properties, individuals and relations between classes and individuals that together form a model of the real world.

As ontologies are altered to correct errors, to accommodate new information or to adjust the representation of the domain, they are said to evolve over time. Hence there is a need for methods and means to manage the ontology evolution in order to ensure that applications using different versions of the ontologies work properly. In this thesis ontology evolution and change are studied by exploring previous research findings and by identifying problems. To enable ontology versioning, an ontology versioning framework is presented and implemented as a set of ontologies and programs. The presented framework aims to provide means for reasoning based on a complete versioning history.

ACM Computing Classification System (CCS):
I.2 Artificial Intelligence,
H. Information Systems,
H.3.6 Library Automation

Avainsanat — Nyckelord — Keywords
ontology, versioning, change

Säilytyspaikka — Förvaringsställe — Where deposited
Library of Department of Computer Science, report code C-2004-

Muita tietoja — övriga uppgifter — Additional information

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Ontologies provide means for explicating concepts of the world and relationships between them. This is done by conceptualizing knowledge about the world into an abstract, simplified view that we wish to use for some purpose. Formally an ontology is an explicit specification of a conceptualization [Gru93]: it thus specifies explicitly a representation of a piece of conceptualized knowledge. In order to do this, ontologies employ notions of class hierarchy and inheritance of properties along a class hierarchy. The subclasses of a superclass inherit properties defined for the superclass. In addition, subclasses may have additional properties.

However, ontologies evolve over time. Ontologies are altered to correct errors, to accommodate new information, or to adjust the representation of the domain as the world changes [HH00]. Hence there is a strong need to revise ontologies. As changes in the ontology are inevitable there must be a strategy for managing a complete ontology version history. There are occasions where differences in the ontology versions disable the ability of an application to work properly. For instance, if the old version of the ontology is used for annotating resources until some time t_1 and a new revised ontology is used after t_1 it is impossible to retrieve all the resources if the new version is not backward-compatible or aligned with the earlier one.

According to research on industrial strength ontology management [DWM01], no current ontology tools support versioning. The tools in the research were Ontolingua/Chimaera [FFR97, MFR+00], Protégé/PROMPT [GEF+99, NM99] and WebOnto/Tadzebao [Dom98]. A strategy, methods, and a systematic implementation of versioning tools are needed to support ontology evolution.

1.2 A Motivating Example

We introduce a motivating example by describing an ontology where changes occur as a function of time. Assume a location ontology that defines different locations and places and their relationships. This ontology describes the state of a simple world during a particular period of time, drawn as a graph. Vertices in the graph describe places and the edges describe a *part of* relation. For example *Finland* is a part of *Europe*. Furthermore, also *Sweden* and *Norway* are parts of *Europe*. Figure 1 depicts some of these relationships in an intuitive way; for example a certain part of *Russia*

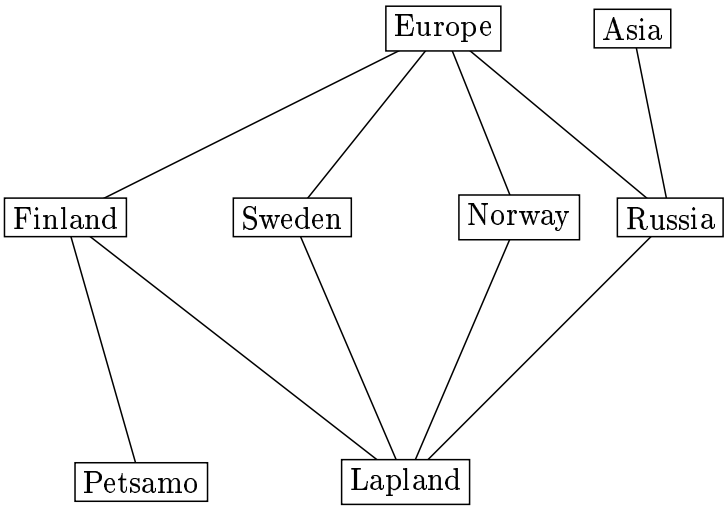

Figure 1: Outline of the "world" before The World War II.

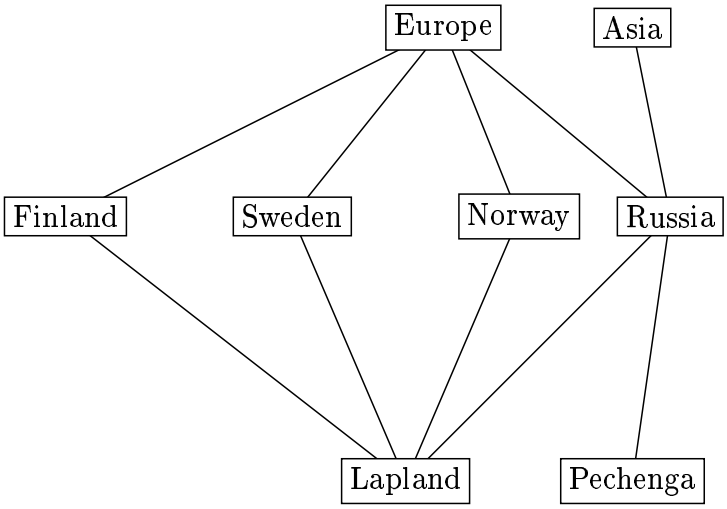

Figure 2: Outline of the modified "world" after The World War II.

is part *Europe* and an other part is part of *Asia*. In the new version in Figure 2 *Petsamo* has the new name *Pechenga* and it is part of *Russia* as a result of the World War II. The question is what should be done with the concept of *Petsamo* that had a part of -relation with the concept of *Finland* in the previous revision. There might be some entities that are annotated using the previous ontology. Assume for example that there is a record in a museum collection about a hat – for example about a "Kolkkahattu" – that has been annotated using the concept *Petsamo* because it has been manufactured in *Petsamo* during the year 1929. If the concept *Petsamo* is deleted, the new ontology cannot be used anymore for querying the old data. The concept of *Russia* is also problematic as its geographical area has been part of *Soviet Union* during a long period of time in the 20th Century.

1.3 The Research Problem

The research problem in this thesis is how to enable reasoning about a complete ontology version history in a web-based context — i.e. what support would be needed for the reasoning to be possible. Recalling that no current ontology editor supports ontology versioning, even a basic set of tools as a framework will help developers and users of an ontology. The main problem is stated as follows:

How to keep track of changes over time of an evolving ontology in order to enable reasoning based on a complete ontology version history?

The research problem is thus a layered, yet a narrow cross-section of ontology versioning, consisting of tightly interrelated areas. Enabling reasoning over ontology version history raises the following subproblem:

How should and can different ontology versions and their evolved elements be represented?

Furthermore, provided that ontology versions and their elements have a unique identifier, another subproblem is:

How can temporal knowledge about concepts be used in inference?

1.4 The Method Chosen

In this thesis the method to solve ontology versioning problems is a combination of different steps. Firstly, ontology versioning is studied by describing and discussing major requirements and the current research results. Furthermore, in order to build a basic framework, the nature of ontology changes are discussed and the consequences of a change are examined and formalized. Finally, an ontology versioning framework is presented and implemented as a set of ontologies and programs that can be used to automate and enable ontology versioning and reasoning using the complete versioning history.

1.5 Organization of This Thesis

In this thesis the problem of ontology versioning is identified, formalized and presented through theory, motivating examples and by exploring previous research findings. Chapter 2 presents ontology development and ontology evolution and discusses

identification, change operations and the role of time: in short how and what should be versioned. To enable ontology versioning, an ontology versioning framework is presented in Chapter 3 and implemented as a set of ontologies and programs presented in Chapter 4.

2 Evolving Ontologies

2.1 Ontology Development and Change

Every single class addition, modification, hierarchy alteration, even the smallest change operation changes the ontology and a new revision of the ontology is formed — and in addition, there can be multiple changes between two revisions. This is of course acceptable since as the picture of what the model of the domain should look like gets clarified, it is natural to try and express it in the ontology as well. An ontology is an explicit specification of a conceptualization [Gru93] and hence if the conceptualization changes, the ontology must also change to match the conceptualized domain.

There are many ways [Kle02] in which the ontology can change. Firstly, the ontology can change silently; the previous version is replaced by the new version without any formal notification and only the new version can be accessed. Secondly, the ontology can change visibly. Then only the new version is accessible; the previous version is replaced by the new version. Moreover, the ontology can again be visibly changed, but now both the new version and the previous version are accessible. Finally, when the ontology is visibly changed, both the new version and the previous version may be in use and are accessible, and there is an explicit specification of the level of compatibility between concepts of the new version and the previous version.

There are several consequences of a change of an ontology O_n:

1. Some *data* conforms to a version Ov_i of the ontology O_n. When the ontology changes and a new version takes place, the data may be interpreted differently or may use unknown terminology.

2. If there are ontologies $O_1, \ldots, O_{n-1}$ that use the changed ontology O_n, then there are *dependencies* between the ontologies [Kor04], and a change may invalidate the dependent ontologies.

3. *Applications* using the ontology O_n might not work with the changed ontology.

4. *Reasoning rules* possibly do not work properly with the new version of the ontology O_n anymore.

In order to solve the above problems, multiple variants — i.e. revisions — of an

ontology must be supported. Creating new revisions is called ontology versioning and it is defined the following way:

Ontology Versioning A method to keep the relation between new revisions of ontologies, the existing revisions of ontologies, and the data that conforms to them consistent [Kle01].

To understand how an ontology evolves, the development process of an ontology must first be examined. It is also necessary to identify different types of ontologies, how likely they are to change, and what forms the changes may take. Developing an ontology includes the following steps [NM01]:

1. *defining classes* in the ontology,
2. *arranging* the classes in a taxonomic, usually subclass-superclass hierarchy,
3. *defining properties* and describing allowed values for these properties,
4. *filling* in the values for properties for instances.

This approach sees the ontology development merely as a waterfall and not as a spiral process. It is likely that an ontology developer, after having made the initial version of the ontology, will at a later phase modify the class hierarchy, fill more values, delete some unsuitable classes, split some classes into separate classes or do some other editing operations he wishes to. A more realistic approach would thus be to see the ontology development process as a continuous work, as ontology evolution. Spiral nature of ontology development cycle and versioning can be combined into the notion of ontology evolution:

Ontology Evolution The ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology [NK03].

This ability to version an ontology consists of methods to distinguish and recognize ontology versions, specifications of relationships between versions, update and change procedures for ontologies, and access mechanisms that combine different versions of an ontology and the corresponding data [NK03]. However, it is not self-evident how ontologies should be versioned, different versions recognized or how the relationships between revisions of ontologies should be specified. Different aspects of version relation between ontologies [KFKO02] include for example:

1. The difference between version relations and conceptual relations inside an ontology.

2. Possible discrepancy between changes in the specification and changes in the conceptualization.

3. Packaging of changes, i.e., the way in which updates are applied to an ontology.

Another approach is to examine differences between relations inside the ontology and relations between versions of the ontology [Kle02]. Ontologies consist of a set of classes c_1, a set of properties p_1 and a set of instances i_1, that is, definitions of them and axioms about them. The classes, properties, instances and axioms are related to each other and together they form a model of a world [Kle02], that is, e.g. a version Ov_1 of the model. A change creates a new version Ov_2 of an ontology O. In addition, a change also creates a version relation between the definitions of classes, properties and instances in the original version Ov_1 of the ontology and those in the new version Ov_2 [Kle02].

The relations between concepts inside an ontology O, e.g. between a class A and a class B, are identified [Kle02] to be fundamentally different from the version relations between two versions of a concept, e.g. between a class $Ov_1.A$ and a class $Ov_2.A$ defined in ontology versions Ov_1 and Ov_2. Relations inside an ontology O are purely conceptual relations in the modeled domain. However, relations between classes defined in two ontology versions Ov_1 and Ov_2 describe meta-information about the change of the concept. Nevertheless, the two versions (e.g. $Ov_1.A$ and $Ov_2.A$) of a concept have a conceptual relation also after a change. This means that although the update relation itself is not a conceptual relation, the participating versions of a concept do have a particular conceptual, that is, a logical relation to each other [Kle02]. It is vital to identify these logical relations between the concepts in order to enable reasoning about an ontology version history.

Moreover, different ontology revisions can have a different status like working draft, final, published and so on. They can also be unofficial versions for testing purposes, that can cause problems. Unless it is possible to explicate whether a revision is official, any agents or query systems that come across the revision will assume that they can use it in place of the old one, and unintended inferences may result [HHL99]. Three methods are suggested [HH00] to prevent this:

1. Agents will only use the revision as a substitute if it only adds categories or

relations. Since such revisions will result in equivalent perspectives for existing data sources, it does not matter if it is an official revision [HH00].

2. A revision must be located on the same server and on the same path as the ontology it revises. This guarantees that the owner has made the revision, but makes it difficult to move the location of an ontology once it has been used [HH00]. Following this approach, e.g. a revision "v1" is found on a server at location http://ontologies.org/musicalontology/v1 and "v2" on the same server http://ontologies.org/musicalontology/v2. Another, slightly different approach is taken by W3C [SWM04]. W3C uses an approach where a name-space includes the time stamp in the front of the ontology name, embedded in the directory structure. Our example would then be http://ontologies.org/2004/03/musicalontology.

3. The original ontology must authorize the revision. This could be accomplished by a *revised-by*-tag that points to the location of the revision. To use this method, upon discovering a purported revision, a system should reload the original ontology and see if it authorizes the revision [HH00].

Furthermore, a possibility to *undo* a change is needed in many cases [SMMS02]. For example, if an ontology engineer fails to understand the consequences of the change that forms a new version Ov_2, an undo-operation restores the original version Ov_1. Undo is also useful when an ontology is changed for experimental purposes and different possible directions of the development are tried. Moreover, when working on an ontology in a distributed environment, ontology engineers may have different ideas about how the ontology should be changed [SMMS02]. An undo-operation enables the reversion to the previous version of the ontology in the case where the new direction is not satisfying the other members of the group.

2.2 How and What to Version

Ontologies are evolving but since there are many domains and different approaches to what the ontology should look like, there must be differences also in the way a change occurs. First of all, changes can be either 1) conceptual changes, where the way a domain is conceptualized changes, or 2) explication changes, where the specification of the conceptualization changes without changing the conceptualization itself [VJBCS97].

Changes also occur differently in different types of ontologies. They are all likely to be revised but not in the same way or with the same frequency. The effect is also different, depending on the type of the ontology. In order to analyze the requirements for a versioning scheme, an analysis on different types of ontologies, including a discussion about the change within them, is provided next.

Domain ontologies capture the knowledge valid for a particular type of domain (e.g. electronic, medical, mechanic, digital domain) [Fen00]. For indexing cultural content there are, for instance, Functional Requirements for Bibliographic Records (FRBR) [Sau98], Categories for the Description of Works of Art (CDWA) [For00], Definition of the CIDOC object-oriented Conceptual Reference Model [Doe03] and IconClass [vdB95]. Domain ontologies can change frequently: the computer industry, for instance, produces new types of computer parts and new combinations of them all the time. If the evolution of the domain is not captured as new revisions of the ontology occur and if the relation to previous versions is not clear, then the maintenance of domain knowledge becomes very hard if not impossible.

Metadata ontologies like the Dublin Core provide a vocabulary for describing the content of on-line information sources [Fen00]. If metadata is widely accepted, describing, for example, terms like author, date, organization, education, and so on, metadata might not change that often. But since these simple metadata ontologies are in some cases also widely used – like Dublin Core is – the change in them has to be given more attention. Furthermore, also changes in the Uniform Resource Identifier (URI) [BLFM98] can cause troubles. For example, the RDF Schema specification [BG00] recommends that "a new namespace URI should be declared whenever an RDF schema is changed". Following this recommendation, the Dublin Core working group changed the URI of their metadata term definition as they published a new version, causing a lot of problems. Later the Dublin Core steering committee has decided to use one URI for all the versions of the Dublin Core metadata set [KF01].

Generic or common sense ontologies aim at capturing general knowledge about the world, providing basic notions and concepts for things like time, space, state, event, process etc. As a consequence, they are valid across several domains. For example, an ontology about mereology (part-of relations) is applicable in many technical domains [Fen00]. Common sense ontologies have the possibility to be used on a really large scale since they provide concepts

for different domains. This means that changes in them can break up massive amounts of applications and other ontologies that are using these generic ontologies.

Representational ontologies do not commit themselves to any particular domain of interest. Such ontologies provide representational entities without stating what should be represented. A well-known representational ontology is the Frame Ontology, which defines concepts such as frames, relations (e.g. one-to-many-relation), properties, and property constraints allowing the expression of knowledge in an object-oriented way [Fen00]. Change in representational ontologies is hard to express. What has really changed if for example a class "slot" is renamed "property"? Changes in representational ontologies can mislead the user for some time and can make it impossible to express certain issues in the new version that were expressible before.

Task and method ontologies provide terms specific for particular tasks (e.g. hypothesis belongs to the diagnosis task ontology) or terms specific to particular methods. Task and method ontologies provide a reasoning point of view on domain knowledge [Fen00].

The above discussion of types of ontologies and analysis of changes serve as a basis for a discussion on the General Thesaurus in Finnish, (Yleinen Asiasanasto, YSA) and its future ontological version, the Finnish General Ontology (Yleinen Suomalainen Ontologia, YSO). According to an interview [YSA03] that was made as a part of this thesis, YSA is a combination of different categories: it aims to cover all domains. It is a generic vocabulary maintained by Helsinki University Library in Finland. YSA contains approximately 14,000 indexing terms (or descriptors) and approximately 3000 non-descriptors. Non-descriptors are synonyms for indexing terms but they are not suggested for use — instead there is a reference from a non-descriptor to the corresponding indexing term. In addition there are currently about 1700 suggestions for new terms to be authorized.

According to the interview, there are currently around 30 new concepts – or in this case words – added each year to the thesaurus but normally no deletions. Modifications of concepts include cases where the form of a descriptor changes, or example, from singular to plural or from plural to singular. Moreover, a literal representation can change, for example, when a fixed phrase is changed to a compound. Furthermore, a non-descriptor can become a descriptor and vice versa. Identified problems

in developing YSA include balancing between different domains and handling of specific terms, short-term phenomena, entrenchment of the form of a term, historical terms and those situations where none of the reference books list the term.

One of the major problems in YSA is the change: how to represent those terms that have been in use for a certain period of time, but that are not used anymore? For example, a fixed-phrase "elektroniset asiakirjat" has been in use until the beginning of year 2004 and it is now deprecated. The term "sähköiset asiakirjat" is a more modern version of this term. Another example is "kuntayhtymät" that has replaced the prior term "kuntainliitot" since year 1993. Currently this temporal information is only given in notes as a free text, which makes machine-based reasoning about temporality of these concepts very hard if not impossible.

Discussions of ontology evolution, version relations and different ontology types usually only concern versioning on the ontology level. Furthermore, versioning implicitly also includes classes, properties, relations and instances described in ontologies. It is important to be able to track movements and changes of concepts and changes between versions and distinguish properties of version relations like the following [Kle02]:

1. *Evolution relation* is the relation that specifies what ontological definition (e.g. a class $Ov_1.A$) in one version is changed into what ontological definition (e.g. a class $Ov_2.A$) in another version.

2. *Transformation* is the actual change as a specification of what has actually changed in an ontological definition, specified by a set of change operations, e.g. change of a restriction on a property, addition of a class, removal of a property.

3. *Conceptual relation* is the relation between constructs in the two versions Ov_1 and Ov_2 of an ontology O. The relations can be specified e.g. by equivalence relations, subsumption relations, or logical rules.

4. *Descriptive metadata* like date, author, and intention of the update describes the when, who and why of the change.

5. *Scope* is a description of the context in which the update is valid. In its simplest form, this can consist of the date when the change occurs in the real world. The date in the descriptive metadata can e.g. be called the transaction

date. More extensive descriptions of the scope, in various degrees of formality, are also possible, like a validity period of a concept.

Due to the nature of ontologies and the wide variety of change types, ontology versioning can be seen as a collection of requirements generated by different viewpoints to the area. This leads to a need to specify the requirements of a versioning and evolution scheme in a more detailed way. The major requirements are identified below.

Identification for every use of a concept or a relation, a versioning framework should provide an unambiguous reference to the intended definition [Kle01]. In other words, ontology metadata is required for providing identifying knowledge about each ontology, such as author, publication date or title. The Dublin Core Metadata Element Set, for example, can be used for the purpose [OWL03].

Change tracking a versioning framework should make the relation of one version of a concept or relation to the other versions of that construct explicit [Kle01].

Transparent translating a versioning framework should — as far as possible — automatically perform conversions from one version to another, to enable transparent access [Kle01].

However, ontologies evolve *over time* and the temporal nature of concepts need to be taken into account. As a result, the list of requirements for ontology versioning should be extended by the following requirement:

Life cycle creation a versioning framework should keep track and help creation of the concept life cycles, that is, temporal intervals expressing validity periods of concepts or relations. This concept life cycle is sometimes called also valid context [KDFO02].

These requirements are discussed in Chapter 3, where an ontology versioning framework is outlined.

2.3 Identification of a Revision

Revisions need to be distinguished and identified in order to provide the identification for every use of an entity. These entities include classes, relations, properties

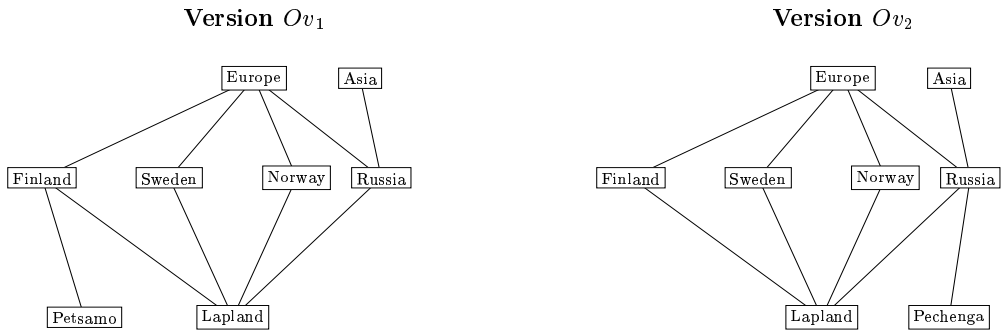

Figure 3: Ontology versions Ov_1 (before the World War II) and Ov_2 (after the World War II).

and instances. For the identification to be clear, a unique reference to the intended definition of the concept is needed [Kle01]. In other words, a reference to a concept in a certain version of the ontology has to be provided with a description about what it is and what version of the ontology is in question, who has altered it and so on. This description can be provided as a set of metadata elements that provide means for tracking down a certain revision of the ontology. In this thesis this metadata is called *revision metadata*. Since description of a resource is a vital question, many different metadata approaches have been suggested. One of the most widely used approach is Dublin Core (DC), which can be used as a basis also when creating the metadata element set for evolving ontologies. Furthermore, OWL (Web Ontology Language) [BvHH+04] provides a few elements for the purpose. Table 1 lists the revision metadata elements selected and developed as a part of this thesis and provides a description for each of them. The source – DC or OWL – is given in parenthesis. The revision metadata elements together form a basis for a Revision Ontology, experimented later in Chapter 4.1.

Metadata can be described either outside the resource or provided as a part of the resource. When discussing ontologies and the requirement of identification the latter seems to be the right choice since when an ontology is published in some distributed network like the World Wide Web (WWW), the identification cannot rely on a centralized metadata server approach. In order to make the ontology be machine-processable, it has to be defined using some formal way. Expressing the metadata by an ontology is discussed in Chapter 4.1.

Another aspect to identification is whether the identification of the ontology also identifies the concepts defined in it. This seems to be a natural way of thinking.

Metadata element	Description
filename	Filename of the ontology
status	Status of the ontology: draft, final or published
author	Author of the ontology (DC)
backwardCompatibleWith	Backward compatible with another version (OWL)
creator	Creator of the ontology (DC)
date	Last modification date of the ontology (DC)
dependsOn	Ontology or ontologies that this ontology depends on
description	Free description of the ontology (DC)
format	Format of the ontology, for example "text/xml" (DC)
identifier	Identifier of the ontology (DC)
incompatibleWith	Incompatible with another version
language	Language of the ontology (DC)
priorVersion	Prior version of the ontology (OWL)
publisher	Publisher of the ontology (DC)
rights	Information about rights held in and over the ontology (DC)
type	Type of the ontology (DC)
relation	A reference to a related ontology (DC)
contributor	An entity responsible for making contributions to the content of the ontology (DC)
coverage	The extent or scope of the content of the ontology (DC)
source	A Reference to a resource from which the present ontology is derived (DC)
subject	Subject of the ontology (DC)
title	Title of the ontology (DC)
versionInfo	Revision number of the ontology (OWL)
versionTag	Version tag of the ontology

Table 1: The Revision Ontology with a description of each element.

If a concept (class, property, relation or instance) is included in the ontology, the identification of the concept can be derived from the ontology identification. Part of the process of identification of a concept is resolving the identity of a concept. If something changes within a concept then the question is how do we decide whether the concept still has the same identity. For example, is "X" the same as "Y", where "X" represents something at one time, and "Y" represents something at a later time? This can be called the problem of change, or the problem of identity over time. The well known Leibniz principle states that

Definition X is the same as Y iff all the properties and relations of X and Y are equal [ME93].

Thus, whatever is true of X is also true of Y, and vice versa. In any imaginable context, an object can be replaced by another, equal, one. The Leibniz principle is strict and disables all possibilities for concepts X and Y to be the same but those defined by the principle. But as a matter of fact, concepts also have another dimensionality in identification. According to research made about ontological changes [NH97], in nearly 98% of the cases the concept that had the same name in different ontology revisions also meant the same thing. If things look similar they probably are such. This means that the whole concept life cycle, running through the different revisions, can be used in the concept identification and in reasoning about concepts. Consequences of this approach are discussed later in Chapter 2.5. Uniform Resource Identifier (URI) has been suggested as the identifier of concepts on the WWW. A URI is a short string of characters, which indicates a name or address that can be used to refer to an abstract or physical resource. According to the original definition [BLFM98], "a resource can be anything that has identity." A URI thus uniquely identifies a resource: if X and Y have the same URI, they are considered to be the same.

2.4 Ontology Changes as Operations

Ontologies evolve when different types of changes occur, and a new revision is created. When a concept in an ontology changes, it always changes in some particular way. Changes occur, for example, when concepts — classes, properties or instances — are added.

These change types can be seen as change operations. A change operation generates some effect as a consequence. For instance, "add a class" inserts a class to the

ontology, "remove an instance" deletes an instance and so on. As operations these are atomic ones and hence not very useful in change-tracking i.e. in making the relation of one version of a concept or relation to other versions of that entity explicit. Thus these operations can be combined to form more covering change-operations such as "merge concepts" or "split concepts". Different change operations can be divided into three categories:

1. *Information-preserving changes* where no instance data is lost. These changes include, for example, creations of classes and properties, attaching properties to classes and adding a subclass-superclass links [NK03]. In other words, if a revision only adds classes, relations, or rules, then it is backward compatible with the original [HFLW02, HH00].

2. *Translatable changes* where no instance data is lost if a part of the data is translated into a new form. Translatable changes include changes like deletions of classes or properties, moving a class up the class hierarchy and re-classifying a class as an instance [NK03].

3. *Information-loss changes* where it cannot be guaranteed that no instance data is lost. In this group changes include, for example, merging classes, splitting classes or encapsulating a set of properties into a new class [NK03].

Translatable changes and information-loss changes are not backward compatible if they remove any components like classes, relations or rules [HH00] and if there are no other means like mappings available that define the situation after the change has occurred and that way enable reasoning about concepts over multiple ontology versions. Different changes have to be detected in order to enable the required change-tracking. There are two main problems with the detection of changes in ontologies [Kle02]:

1. The abstraction level at which changes should be detected. Abstraction is necessary to distinguish between changes in the representation that affect the meaning, and those that do not influence the meaning. It is likely that the same ontological definition is represented in different ways. Thus tracking the changes in the representation alone is not sufficient.

2. Even when the correct level of abstraction is found for change detection, the conceptual implication of such a change is not yet clear. Because of the difference between conceptual changes and explication changes, it is not possible to

derive the conceptual consequence of a change completely on the basis of the visible change only (i.e. the changes in the definitions of concepts and properties). Heuristics can be used to suggest conceptual consequences, but the intention of the ontology designer determines the actual conceptual relation between versions of concepts.

2.5 Change as a Function of Time

Ontologies change over *time*, meaning that time has to be given a significant role when discussing an ontology evolution. For instance, knowing that Petsamo used to belong to Finland, the sentence "Petsamo became part of Soviet Union September 19, 1944" implies that "Petsamo" was a part of "Finland" September 18, 1944 and that "Petsamo" was a part of "Soviet Union(SU)" September 19, 1944. Formally this can be expressed as follows:

$$\text{PartOfFinland}(\text{“Petsamo”},1944/09/18) \wedge \neg\ \text{PartOfFinland}(\text{“Petsamo”},1944/09/19) \wedge \neg\ \text{PartOfSU}(\text{“Petsamo”},1944/09/18) \wedge \text{PartOfSU}(\text{“Petsamo”},1944/09/19)$$

There are different theories about what time really is. One of the widely used theories in the area of artificial intelligence says that history is linear and the future is branching [Woo00]. But since ontology versioning is here considered as a problem of handling past changes, only the notion of the linear history is needed. Thus no predictions of future changes are made. Furthermore, linear history assumes that ontology versioning has happened as a series of changes that always change the previous version of the ontology O and thus no branching occurs in the version history.

In order for this linear time to be machine-processable, it has to be modeled as temporal entities to enable reasoning about time-enriched ontologies. As one solution, the XML Schema provides [BM01] built-in primitive datatypes that relate to time. They are dateTime, duration, time, date, gMonth, gMonthDay, gDay, gYear and gYearMonth. XML Schema datatype duration is a standard for expressing the duration of an interval, and it is defined [BM01] as follows:

Duration represents a duration of time. The value space of duration is a six-dimensional space where the coordinates designate the Gregorian year, month, day, hour, minute, and second components defined in § 5.5.3.2 of [ISO88],

respectively. These components are ordered in their significance by their order of appearance i.e. as year, month, day, hour, minute, and second.

Moreover, The Suggested Upper Merged Ontology (SUMO) [NP03] suggests the following concepts of time that includes more entities than XML Schema:

Classes April, August, day, December, February, friday, hour, January, July, June, leap year, March, May, minute, monday, month, November, October, saturday, second, September, sunday, thursday, tuesday, wednesday, week, year

Instances negative infinity, positive infinity

Relations before, before or equal, cooccur, date, duration, during, earlier, finishes, frequency, meets temporally, overlaps temporally, starts, temporal part, temporally between, temporally between or equal

Functions begin, day, end, future, hour, immediate future, immediate past, minute, month, past, recurrent time interval, second, temporal composition, time interval, when, year

The relations suggested by SUMO have many concepts that are equal to the ones proposed in the Allen Algebra [All83]. In the Allen Algebra an initial structure is aRb where R denotes the set of the 13 primitive interval relations that exclusively correspond to every possible simple qualitative relationship that may exist between a pair of intervals (see Figure 4).

Allen Algebra enables one way to reason about a complete version history, for example, by examining the truth value of (ValidityPeriod(class A) `before` ValidityPeriod(class B)), where ValidityPeriod() returns a temporal interval assigned to a class. In the example, defining a location ontology in Figure 5 we would get, for example the following facts. Notice that many of these facts can be inferred from the other facts too.

(ValidityPeriod(*Petsamo*) `before` ValidityPeriod(*Pechenga*)) = `true`
(ValidityPeriod(*Pechenga*) `after` ValidityPeriod(*Petsamo*)) = `true`
(ValidityPeriod(*Petsamo*) `equals` ValidityPeriod(*Pechenga*)) = `false`
(ValidityPeriod(*Petsamo*) `meets` ValidityPeriod(*Pechenga*)) = `true`
(ValidityPeriod(*Pechenga*) `met-by` ValidityPeriod(*Petsamo*)) = `true`
(ValidityPeriod(*Petsamo*) `overlaps` ValidityPeriod(*Pechenga*)) = `false`

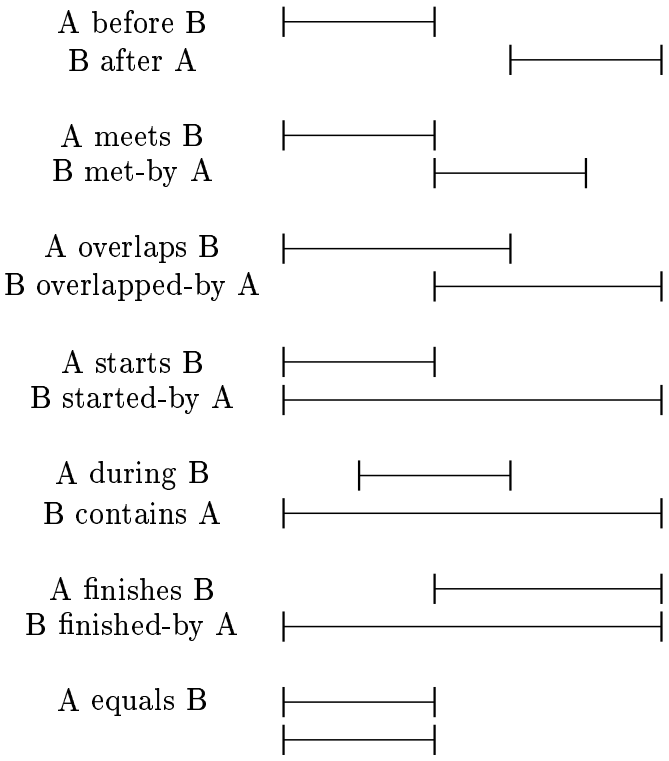

Figure 4: Time interval relations in the Allen Algebra [All83].

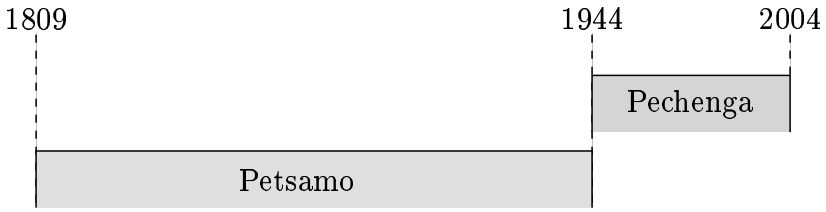

Figure 5: A class *Petsamo* is valid before an other class *Pechenga*.

Classes of SUMO seem to cover the basic concepts of time well, like months, weekdays and other notions of periods. However, in ontology versioning we are interested in concept life cycles, that is, periods of time during which a concept has been valid, its valid context. We need to know exactly when the concept has been added into the ontology and if it has at some time point been taken away or modified using some ontology change operation. Another choice is that we at a later phase want to give a concept a certain validity period when modeling historical facts. Moreover, it would be useful to be able to provide names for the periods in order to form more descriptive events. One example of such an event is the period of time consisting of when the fact "Finland `partOf` Russia" was valid. The life cycle of this `partOf` relation between "Finland" and "Russia" holds from 1809 until 1917.

This life cycle can be given a status of an event and named for example as "Finland `partOf` Russia". All this suggests a time ontology that can be used to express different temporal entities. A recent suggestion [PH04] provides this kind of ontology, defining basic temporal concepts like Instant, Interval, Instant Event, and Interval Event, depicted in Figure 6 as most specialized classes of a subclass hierarchy of temporal concepts. Instants are time points and intervals are things with extent, that is, they have a start time and an end time. Here instant events are events that are instantaneous, such as Finland joining the European Union or arrival of a letter. Furthermore, interval events are events that span some time interval, and that have also a description like "Finland `partOf` Russia". There are also five other more general temporal concepts or classes: Temporal Thing, Temporal Entity, Instant Thing, Interval Thing, and Event that form the upper part of the class hierarchy. Time information can be used in ontology versioning. For example, concepts in two versions Ov_1 and Ov_2 of a location ontology can be given validity periods, like depicted in Figure 7. Here a relation between the concepts *Petsamo* and *Pechenga* is not yet expressed; this will be discussed later in Chapter 3.1.

Concepts of an ontology have to be bound with time. There are at least three choices of how to add temporal knowledge to concepts:

1. Add relations between classes and their temporal intervals — life cycles or validity periods — by hand. This is needed especially in modeling historical facts.

2. Use some automation method to automatically generate temporal relations, based e.g. in version history information given by the version control system used.

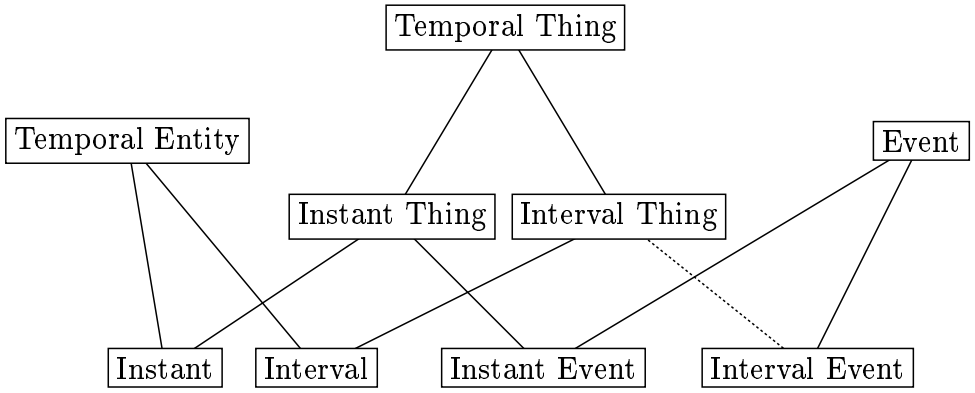

Figure 6: Subclass hierarchy of temporal concepts [PH04].

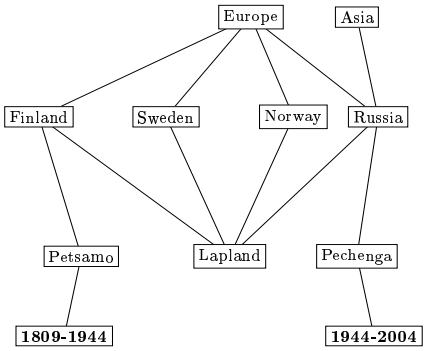

Figure 7: Combination of ontology versions Ov_1 and Ov_2 where validity periods of Petsamo and Pechenga are expressed using time intervals.

3. A combination of 1 and 2. Here the automation method 2 suggests validity periods for concepts; they can be adjusted later by hand.

2.6 Automatic Evolution Tracking

In order to fulfill the need for reasoning about a complete ontology version history, a versioning framework must use maximal automation to perform conversions from one version of the ontology to another version in order to enable transparent access. Recall that ontology-change operations were presented and discussed before in Chapter 2.4. An approach to translatability is taken by examining what possibilities we have to track down the changes. The different approaches are called traced evolution and untraced evolution [NK03] and they are defined as follows:

Traced evolution Evolution is treated as a series of changes in the ontology. After each operation that changes the ontology (e.g. add or delete a class, attach a property to a class, change restrictions on properties, etc.), we consider the effects on the instance data and related ontologies, depending on the dimension of compatibility we use. The resulting effect is determined by the combination of change operations. Traced evolution produces logs that can be examined to get a picture of what has changed [NK03]. In other words, traced evolution is successful change tracking, in the sense that it provides explicit relations between one version of a concept (or relation) to the other versions of that construct. This has been identified as one of the main requirements [Kle01] of an ontology versioning framework.

Untraced evolution All we have are two versions of an ontology and no knowledge of the steps that led from one version to another. We will need to find the differences between the two versions in an automated or semi-automated way [NK03] in order to track what has changed in the ontology.

The problem of comparing ontology versions Ov_1 and Ov_2 is more simple if there are change logs available, like in traced evolution [NM03]. However, if the ontology development is distributed like, for example, in the planned General Finnish Ontology (YSO) development, the traced evolution with logging capabilities is not a realistic approach [NM03]. Furthermore, ontology-development tools do not always support logging.

If logs cannot be used in many cases there have to be other means to help in translating between ontology versions. One solution is to ensure that the new version is backward-compatible with the old version. Backward compatibility of an ontology O can be formalized in the following way [Hef01]:

Definition An ontology Ov_2 is backwards compatible with an ontology Ov_1 iff every intended model of Ov_1 is an intended model of Ov_2 and $V_1 \subseteq V_2$, where Ov_1 and Ov_2 are two revisions of the ontology O and V_1 and V_2 are their vocabularies, accordingly.

This simply means that if every logical consequence of the original is also expressible in and a consequence of the revision, then the revision is backward compatible [KF01]. The semantics of an ontology are changed in backward compatible versions in such a way that the interpretation of data via the new version Ov_2 is the same as when using the previous version Ov_1 of the ontology. Backward compatibility is also transitive: if ontology version Ov_2 is backward compatible with version Ov_1 then also the latest version Ov_3 is backward compatible with version Ov_1 [KF01]. Analyzes have shown that if a revision only adds categories, relations, or rules, then it is backward compatible with the original, whereas if it removes any of these components, then it is not backward compatible [HFLW02, HH00]. Other dimensions that we must consider when determining whether a new version of an ontology is compatible with the old one are [NK03]:

1. *Instance-data preservation* — no data is lost in transformations from the old version to the new one.

2. *Consequence preservation* — if an ontology is treated as a set of axioms, all the facts that could be inferred from the old version can still be inferred from the new version.

3. *Ontology preservation* — a query result obtained using the new version is a superset of the result of the same query obtained using the old version. Segregating consequence and ontology preservation is artificial though, because answering to a query is a special case of the inference defined in the consequence preservation.

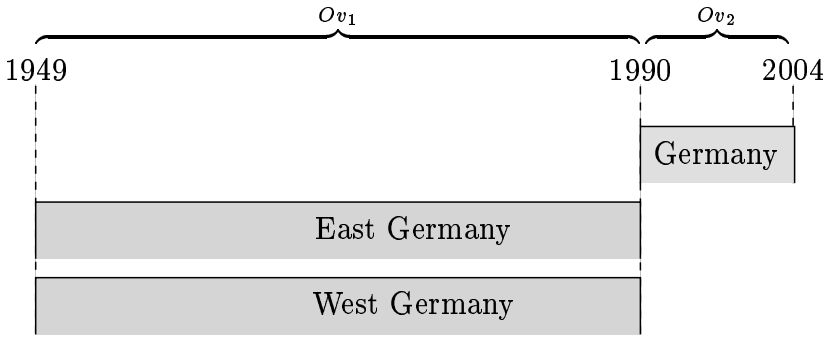

Figure 8: Classes *West Germany*, *East Germany* and *Germany* from ontology versions Ov_1 and Ov_2.

3 A Framework for Ontology Evolution

3.1 Bridging the Gap between Ontology Revisions

The ontology change operations presented in Chapter 2.4 are only half-way towards the requirement of actual change-tracking, not to mention the requirement of translating between revisions of the ontology. Assume that when the ontology changes, the old material is still using the old ontology Ov_1 to describe the content, and the new material is using the new ontology Ov_2. This means that different versions of the same ontology may be incompatible after the change has been implemented and the new version taken into use. The application has to be told explicitly which version of the ontology it is accessing and how it has changed from one version to another so that it can perform accordingly [DWM01].

Recall the definition of backward compatibility presented in Chapter 2.6. Let us assume two versions Ov_1 and Ov_2 of a location ontology defining concepts *Germany*, *East Germany* and *West Germany*, depicted in Figure 8. The new version Ov_2 of the ontology is not backward-compatible with the previous version Ov_1 since classes "East Germany" and "West Germany" are no longer part of the ontology.

Moreover, the old version Ov_1 is not forward-compatible either, since class "Germany" is not present. However, it would be essential to use multiple versions of the ontology in reasoning about the resources, independently of the ontology changes and independently of the formal backward-compatibility. In our example, the user might for example be interested in those resources – say for example German wines — that were before annotated with either the class "East Germany" or "West Ger-

many" and are nowadays annotated with the class "Germany". Since there are a lot of different revising needs for an ontology — like correcting errors, accommodating new information and adjusting the representation of a particular domain [HH00] —, it is not likely that the requirement for backward-compatibility is easy or even possible to always meet. Thus other means, such as ontology mapping, are needed to enable reasoning about an evolved ontology.

Mappings between ontologies have previously been discussed [MWK00], but mappings between ontology versions need a different approach. These revision mappings have to be identified and used to create bridges between those classes (or properties or instances) from ontology revisions Ov_1 and Ov_2 that the change has touched. Let us call this mapping a *change bridge* and define it the following way:

Change Bridge is a mapping between (i) a set of classes c_1, a set of properties p_1 and a set of instances i_1 from ontology Ov_1 and (ii) a set of classes c_2, a set of properties p_2 and a set of instances i_2 from an ontology Ov_2 such that it defines an one-to-one, an one-to-many or a many-to-one relation between the entities that a change concerns such that the change bridge describes the relation between entities after the change has occurred.

In other words, one bridge can help multiple entities from ontology versions to meet each other again, but it does not have to express all the changes at once. If there are many changes between two versions of an ontology, multiple change bridges can be used to express all of them. In an ontology evolution of a location ontology, for instance, semantic gap between different versions Ov_1 and Ov_2 can be bridged like depicted in Figure 9, where two ontology versions Ov_1 and Ov_2 are combined and the change is expressed as an instance of a change bridge called *usedToBe*.

In order to help identifying possible bridges and how they should be used, the following set of relevant questions are to be answered:

1. How can different ontology versions be identified?
2. What has changed (in the old version Ov_1)?
3. What has it changed into (in the new version Ov_2)?
4. How can the change be explicitly expressed, stored and maintained as a set S_{CB} of change bridges CB between the classes of versions Ov_1 and Ov_2 of the ontology?

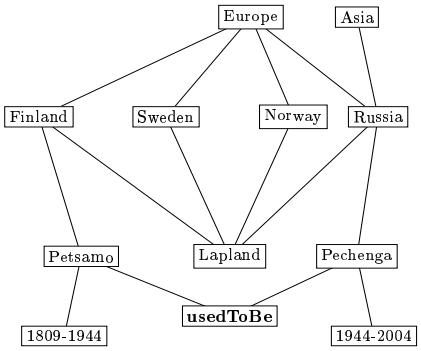

Figure 9: Combination of ontology versions Ov_1 and Ov_2 where the relation between Petsamo and Pechenga is expressed using a Change Bridge.

5. How can the whole ontology version history be used to reason about the relations between concepts (i.e. how the resources can be found when different versions have been used)?

Change bridges can be expressed as a change-bridging ontology — discussed in the next Chapter 3.2 — that provides a set of classes for expressing the situation *after* the change has occurred. This expression is in simple cases a mapping between one or more classes found in version Ov_1 and one or more classes of version Ov_2 of the ontology. In a more complex situation, the counterparts can also be instances of the classes or properties of the classes.

3.2 Expressing Change Bridges

There are basically two choices how change bridges could be expressed in evolved ontologies. They are:

1. Change-Bridging Ontology which mediates and expresses the changes between versions Ov_1 and Ov_2 of the ontology. This is similar to the previous approach [MWK00] for mappings between source ontologies where ontologies remain as separate entities and certain articulation rules provide the mapping between ontologies.

2. Direct mappings between versions Ov_1 and Ov_2 of the ontology.

The difference between these two approaches is depicted in Figures 10 and 11. Where the mediating ontology does not touch the versions Ov_1 and Ov_2 of the ontology at

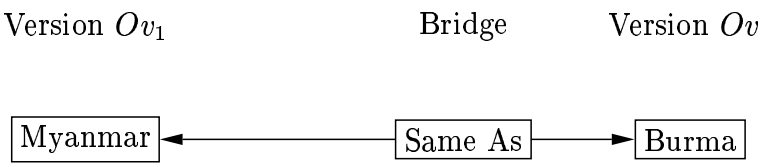

Figure 10: Example of a Same As -relation between two ontology versions Ov_1 and Ov_2.

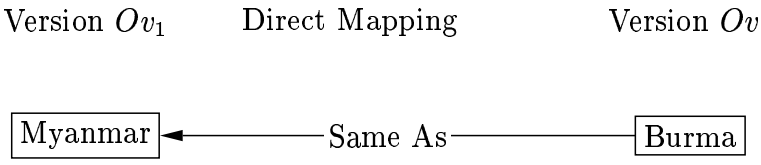

Figure 11: Example of a direct mapping between ontologies.

all but instead points to the entities in them, the direct mapping approach requires additional properties to at least one of the versions Ov_1 or Ov_2. In direct mapping, the “Same As”-relation is expressed as a directed edge between classes “Myanmar” and “Burma” from two ontology versions Ov_1 and Ov_2. This means that “Same As” is a property of class “Burma”. This is a problematic approach since altering the original ontology revisions is in many cases impossible.

To solve this, in a mediating, change-bridging ontology approach, the “Same As”-relation is expressed outside the original ontologies. If the original versions are kept as untouched as possible then the first choice has to be selected, the mediating ontology approach.

A simple example illustrates the difference between these approaches: assume still that "East Germany" and "West Germany" have been classes of the ontology version Ov_1 and that they have both been deleted from the ontology to form Ov_2 of the ontology where also a new class "Germany" is added at the same time. Whereas a change operation only describes that an operation (e.g. deletion or addition op-

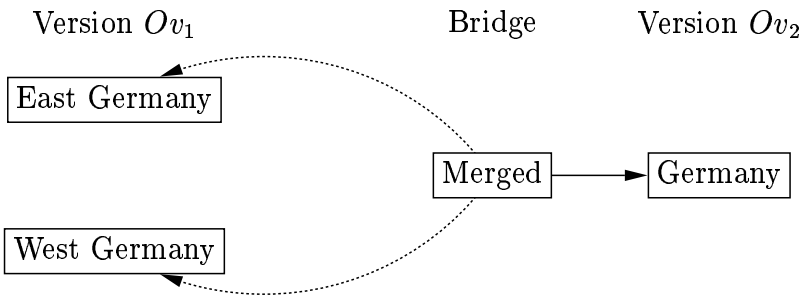

Figure 12: Usage of merged-relation from the Change-Bridging Ontology.

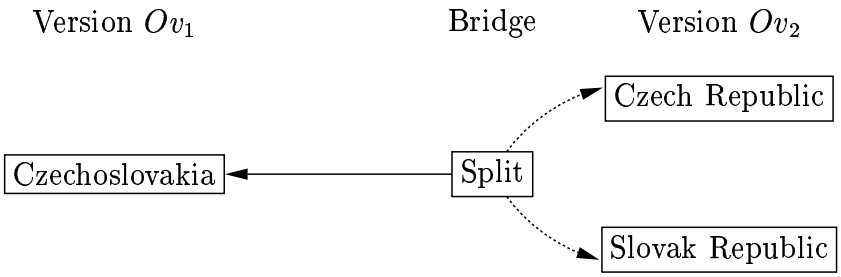

Figure 13: Usage of split-relation from the Change-Bridging Ontology.

eration) *has occurred* a bridging class describes the *relation* between the set S_1 of deleted classes ("East Germany" and "West Germany") and the set S_2 of added classes (consisting here of only one class, namely "Germany").

In this case the mapping relation would be "merged" as classes "East Germany" and "West Germany" have been merged to one single class "Germany" in the new version. In another words, in version Ov_1 of the ontology *East Germany* and *West Germany* are separate concepts and in version Ov_2 they are merged to form *Germany*. Figure 12 depicts the classes and a newly created relation, namely the `merged` relation. Notable here is the direction of the arrows: the ontology versions Ov_1 and Ov_2 do not have to be touched since the directed edges are formed from the bridging class (and not the other way round).

To give another illustration, Figure 13 presents another usage example of a bridging class, a `Split`-relation. Here class "Czechoslovakia" has been split into two distinct classes "Czech Republic" and "Slovak Republic" in the revised ontology. Again the bridge — this time a `Split`-relation — describes the newly formed relation.

The classes of the Change-Bridging Ontology are presented in Table 2. This Change-Bridging Ontology and the classes that are found in it are a formalization and an application of different change operation types presented in Chapter 2.4. Contrary to the change operations, the Change-Bridging Ontology does not try to express change operations — such as deletion of a class, addition of a class, and so on — themselves, but rather the relations between the revised classes of versions Ov_1 and Ov_2 of the ontology.

In other words, a clear separation between the change operations and the explicit relations between the classes has to be made for the mapping to be effective and most expressive. Different bridge classes form a simple subclass-hierarchy depicting the different roles and types of bridges they have and inherit. In short, the upper level classes are defined as follows:

classChange is a bridge intended to be used when something has happened in the class level between classes of versions Ov_1 and Ov_2 of the ontology. Typical bridges of this kind are merged, split and sameAs. To form these bridges we have to know which classes are present in versions Ov_1 and Ov_2 or merely the differences between classes in Ov_1 and those in Ov_2.

hierarchyChange combines different bridges defining the manipulation of the class hierarchy. In other words, in this case versions Ov_1 and Ov_2 of the ontology have a difference in their hierarchies but they can still have exactly the same classes in them. Typically, hierarchy alteration occurs when classes or properties are moved down or up in the hierarchy or subclass-superclass relations are modified.

propertyChange describes the situation where some properties of classes of Ov_1 have been altered to form a new version Ov_2 of the ontology. A typical example of a property change is the `samePropertyAs`-relation defining the similarity of two distinct properties.

typeChange is a crucial bridge, defining a mapping between those classes and instances of Ov_1 and classes and instances of Ov_2 where a re-classification (class to instance or vice versa) has occurred.

Consequently, simple usage rules for Change-Bridging Ontology are needed, describing how it should and can be used to form mappings between versions Ov_1 and Ov_2 of the ontology. These rules include:

Change type	**Relation after change**
classChange	classesDeclaredDisjoint
	differentFrom
	merged
	sameAs
	split
	usedToBe
hierarchyChange	classMovedDown
	classMovedUp
	propertyMovedDown
	propertyMovedUp
	subclassSuperclassLinkAdded
	subclassSuperclassLinkRemoved
propertyChange	narrowedPropertyRestriction
	samePropertyAs
	widenedPropertyRestriction
typeChange	classRe-classifiedAsInstance
	instanceRe-classifiedAsClass
	setOfPropertiesEncapsulatedIntoNewClass

Table 2: Concepts of a Change-Bridging Ontology.

1. Relations between versions Ov_1 and Ov_2 are expressed using the Change-Bridging Ontology by creating instances of the classes it suggests.

2. The mapping is stored preferably in a separate file.

3. The arrows (directed edges) point *from* the bridge classes *to* the certain classes of versions Ov_1 and Ov_2 of the ontology.

4. Mappings can be made either between the entities of versions Ov_1 and Ov_2 of the ontology or between the entities found only in Ov_2. In other words, the ontology modeler can leave — if he chooses so — the outdated classes also to the new version and provide bridges with the more recent classes.

5. When a mapping is made, it has to be complete, that is, no halfway bridges having only partial information are allowed.

6. A revision ontology is used to automatically get identification, status, author

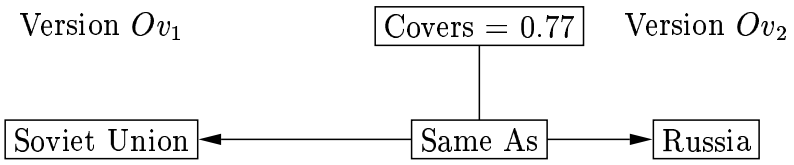

Figure 14: The territories of Russia and Soviet Union are used for calculating the "covers" property.

> and other important identification information from the versioning system in use

When using a Change Bridge like sameAs it is assumed that the concepts are completely the same. There are situations, however, where a sameAs relation after the change has occurred is only partial. Consider for example classes "Soviet Union" and "Russia". In a location ontology it would be reasonable to have a sameAs-relation between the deprecated class "Soviet Union" in an old ontology revision Ov_1 and the class "Russia" of a new ontology revision Ov_2. The problem is that these classes are partly different, e.g. the countries they represent differ by size, the administration is different and so on. For these reasons the definition of the change bridge could be extended to include a "covers" property that gets values between 0 and 1. Calculating a value for "covers" property always needs a viewpoint. For example, here, the "covers" property of relation (Russia `sameAs` SovietUnion) is calculated by comparing the territories modern day Russia occupies with the territories Soviet Union used to occupy i.e. how much the territory of Russia covers the territory of former Soviet Union. Thus, we calculate

$$\frac{Territory(Russia)}{Territory(SovietUnion)} = \frac{17{,}075{,}200 sq.km}{22{,}274{,}900 sq.km} \approx 0.77$$

Figure 14 depicts the new situation. Other possibility would be, for example, to calculate the overlap using some other property such as the number of inhabitants, length of land boundaries or the coastline, depending on the selected viewpoint.

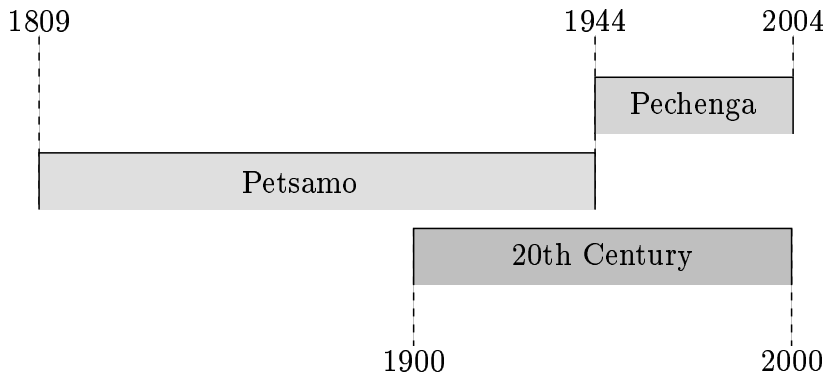

Figure 15: Temporal validity (or persistence) of the classes *Petsamo*, *Pechenga* and *20th Century*.

3.3 Temporal Overlap in Sorting Query Results

Ontology classes and instances with temporal information can be used to sort the query results. For example, assume that classes *Petsamo* and *Pechenga* have been valid in different revisions of an ontology during the periods listed in Figure 15. Recall the motivating example from Chapter 1.2 defining an ontology where *Petsamo* is a part of *Finland* in the ontology version Ov_1 and in the new version Ov_2 *Petsamo* has the new name *Pechenga* and it is a part of the *Soviet Union* as a result of the World War II. Assume also that there is a record in a museum collection about a hat — for example a "Kolkkahattu" — that has been annotated using the time interval concept *20th Century* and moreover using both the concepts *Petsamo* and *Pechenga*. Assume also that resources are evenly distributed over the temporal intervals of the concepts used for annotating them.

We are interested in providing the user with a probability of whether the hat is from *Petsamo* or from *Pechenga*, using only the given annotation knowledge. If we query the ontology using the time period *20th century*, the query results can be sorted using the validity periods of *Petsamo* and *Pechenga* depending how much they overlap *20th century*. Temporal overlap of concepts that have some relation between them give an estimate about how relevant concepts are to each other.

(i)

Following our example, the probability that the hat is from *Pechenga*, when the user searches a hat of the *20th Century* is

$$\mathrm{P}\left[V_1 \mid T\right] \;=\; \frac{\mid V_1 \cap T \mid}{\mid T \mid},$$

where V_1 = validity period of *Pechenga*, valid during an interval from 1944 until 2004 and T = *20th Century*, meaning years from 1900 until the end of year 1999.

$\mathrm{P}[V_1 \mid T]$ is calculated as follows:

$$\begin{aligned} \mathrm{P}[V_1 \mid T] &= \frac{\mid V_1 \cap T \mid}{\mid T \mid} \\ &= \frac{2000 - 1944}{2000 - 1900} \\ &= \frac{56}{100} \\ &= 0.56 \end{aligned}$$

In a similar way, we get the probability that the hat is from *Pechenga*, when the user searches a hat of the *20th Century*. We get

$$\mathrm{P}[V_2 \mid T] = \frac{\mid V_2 \cap T \mid}{\mid T \mid}$$

where V_2 = validity period of *Petsamo*, valid during an interval from 1809 until 1944 and again T = *20th Century*.

Next we calculate $\mathrm{P}[V_2 \mid T]$:

$$\begin{aligned} \mathrm{P}[V_2 \mid T] &= \frac{\mid V_2 \cap T \mid}{\mid T \mid} \\ &= \frac{1944 - 1900}{2000 - 1900} \\ &= \frac{44}{100} \\ &= 0.44 \end{aligned}$$

(ii)

Next, assume that there are means such as change bridges defined before in Chapter 3.1 to provide mappings between ontology revisions. Assume that the classes *Petsamo* and *Pechenga* have been valid in different revisions of the ontology during the same periods and that a change bridge CB, for example `usedToBe`, has been formed between classes *Petsamo* and *Pechenga* because they represent the same geographical areas. Hence we get *Pechenga* `usedToBe` *Petsamo*. Let us assume again

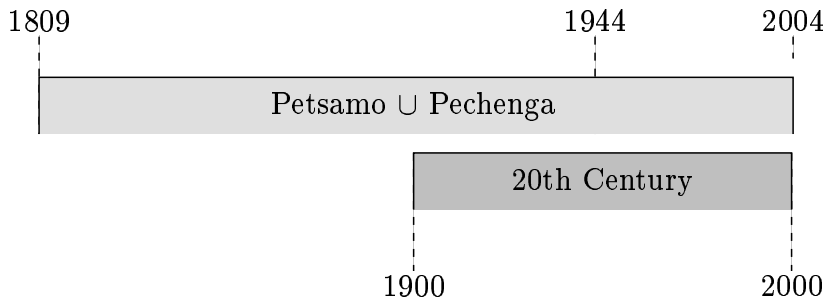

Figure 16: Temporal validity of classes *Petsamo*, *Pechenga* and *20th Century*.

the record about a hat that has been annotated using a time interval *20th Century* and using both the concepts *Petsamo* and *Pechenga*. A combined temporal period *Petsamo* ∪ *Pechenga* is from 1809 to 2004. We are interested in providing the user the probability, that tells whether a hat of our example is from *Petsamo* or from *Pechenga*, using only the given annotation knowledge. If we query the ontology using the time period *20th century*, the query results can be sorted using the validity periods of *Petsamo* and *Pechenga* depending how much they overlap *20th century*.

We get the probability that the hat is from an area called previously *Petsamo* and nowadays *Pechenga*:

$$\mathrm{P}\left[V_3 \mid T\right] \;=\; \frac{\mid V_3 \cap T \mid}{\mid T \mid}$$

where V_3 = validity period of *Petsamo* ∪ *Pechenga*, valid during an interval from 1809 until 2004 and again T = *20th Century*.

We can calculate $\mathrm{P}\left[V_3 \mid T\right]$:

$$\begin{aligned}
\mathrm{P}\left[V_3 \mid T\right] &= \frac{\mid V_3 \cap T \mid}{\mid T \mid} \\
&= \frac{2000 - 1900}{2000 - 1900} \\
&= \frac{100}{100} \\
&= 1.0
\end{aligned}$$

4 The Framework Implemented and Experimented

4.1 A Revision Ontology for Versioning

The two most used knowledge models that can be used in implementing ontologies are RDF (Resource Description Framework) [LS99] and a more recent extension, OWL (Web Ontology Language) [BvHH+04]. Now if ontologies are defined, say, in OWL, and if the revision metadata has to be incorporated into the ontology in order to meet the requirement for ontology identification it sounds natural to describe also the metadata using OWL. In this way existing means for including other ontologies — like the import-directive of OWL — can be taken into use. The ontological metadata can hence be imported (included) into the main ontology and the identification requirement met. Below is a short example of usage of the properties description, format and identifier of Revision Ontology developed as a part of this thesis:.

```
<rdf:Property rdf:about="&Versioning;description"
         a:defaultValues="Free description of the ontology"
         a:maxCardinality="1"
 rdfs:label="description";
         rdfs:comment="This ontology is intended for describing classes and
                     relations of musical works">
        <rdfs:domain rdf:resource="&Versioning;Versioning"/>
        <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&Versioning;identifier"
         a:defaultValues="$Id: Versioning.rdfs,
                          v 1.4 2004/03/01 13:36:12
                          tomikauppinen Exp $"
         a:maxCardinality="1"
         rdfs:label="identifier">
        <rdfs:domain rdf:resource="&Versioning;Versioning"/>
        <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
```

Furthermore, the recently proposed Ontology Web Language (OWL) annotation properties [BvHH+04] can be used in providing metadata, like the following short example shows:

```
<owl:AnnotationProperty rdf:about="&dc;creator"/>
<owl:Class rdf:about="TheMiraculousMandarin">
  <rdfs:label>The Miraculous Mandarin</rdfs:label>
  <dc:creator>Béla Bartók</dc:creator>
</owl:Class>
```

Even though metadata is important in identifying an ontology, creating metadata requires a lot of error-prone handwork and hence the metadata generation phase should be automated as much as possible. Although ontology versioning is still in its infancy, good revising practices have been created in the area of software development. A good version control system, such as CVS (Concurrent Versions System) [Ber90], provides the basic versioning automatically, including keeping a log of when, and why changes occurred and who conducted them.

CVS also provides means to include version identification information in the handled files through usage of a special set of CVS keywords. Every time the resource is committed to the version management system, the system fills up the latest information inside the special tags. Thus, using the keywords we can now automatically provide contents for many of the metadata elements described above. This is a big advantage in the identification of ontology elements. Furthermore, it can be used in reasoning about complete ontology evolution. Those ontology elements in which the keywords can be used are listed in Table 3.

filename	$RCSfile: Versioning.rdfs,v $
state	$State: Exp $
author	$Author: tkauppin $
creator	$Author: tkauppin $
date	$Date: 2004/03/01 13:36:12 $
identifier	$Id: Versioning.rdfs,v 1.4 2004/03/01 13:36:12 tkauppin Exp $
versionInfo	$Revision: 1.4 $
versionTag	$Name: $

Table 3: Those metadata elements of the Revision Ontology that can be filled automatically by CVS.

4.2 Difference Tracking and Change Bridge Generation

Another possibility to aid reasoning about versions is to produce change bridges (i.e. mappings) as was introduced in Chapter 3.2. Forming the change bridges between the ontology classes from version Ov_1 and version Ov_2 of the ontology O must be made as automatically as possible. Algorithm 1 outlines the method used in the implementation that is made as a part of this thesis. Comparison of two ontology versions is not straightforward. For instance, the RDF Schema specification [BG00] recommends that "a new namespace URI should be declared whenever an RDF schema is changed". If the recommendation is followed and a new namespace for each version created $(Ov_1, Ov_2, Ov_3, \ldots)$ then all the classes and instances are considered different. Thus when comparing ontology versions, a common namespace — at least a temporal one — for all the versions to be handled is needed.

Furthermore, it would be ideal if the system could further narrow down the possible selections and suggest certain relations automatically, like illustrated in Figure 17, where the most likely bridge, "merged", is in bold.

In previous research findings about ontologies a method is developed to facilitate defining rules that link different ontologies [MWK00]; these rules are called graph transformation rules. A similar approach would possibly benefit ontology versioning research as well but it is not researched as a part of this thesis.

v1 class	suggested bridge	v2 class
http://ontologies/#EastGermany	sameAs	http://ontologies/#Germany
http://ontologies/#WestGermany	split	
	merged	
	differentFrom	
	usedToBe	

Figure 17: Output of the *version bridger*.

```
input  : Version Ov1 and version Ov2 of the ontology O
output: Change bridges between the ontology classes
begin
    //track how version 1 differs from version 2;
    diff1 = version1.difference(version2);
    //
    //track how version 2 differs from version 1;
    diff2 = version2.difference(version1);
    //
    while differences left do
        changeBridges.add(getSuggestion(diff1,diff2));
    end
end
//Return the change bridges as an ontology ;
return changeBridges;
```

Algorithm 1: Algorithm for checking the differences and forming the change bridges. Suggestions can be either given by the user or generated automatically.

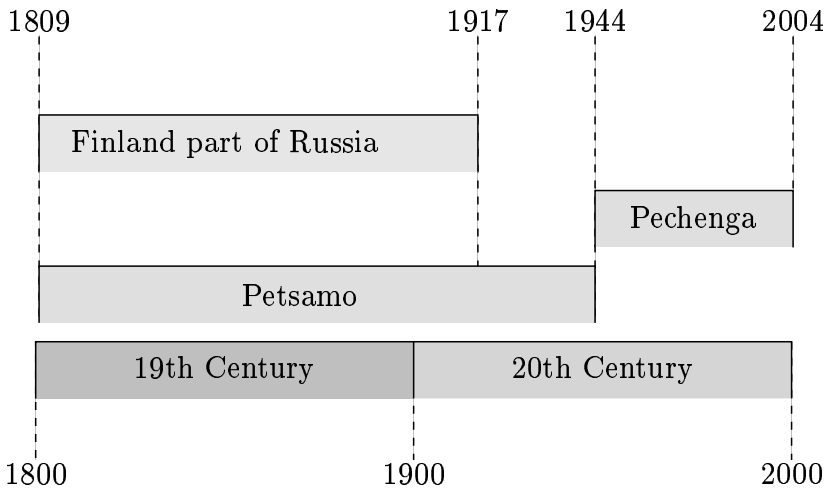

Figure 18: An example of temporal intervals.

4.3 Handling Time and Sorting Query Results

A temporal ontology — or Time Ontology — can be used to instantiate different periods of time, like the validity periods of classes depicted in Figure 18. As a part of this project the Time Ontology has been developed using RDF, adopting the approach presented by Pan and Hobbs [PH04]. Below is a short example of how a period can be constructed as an instance to form descriptive temporal events: it defines the period of time consisting of when the fact "Finland `partOf` Russia" was valid.

```
<TimeOntology:IntervalEvent rdf:about="&TimeOntology;TO1"
        TimeOntology:StartTime="1809"
        TimeOntology:EndTime="1917"
        rdfs:label="Finland partOf Russia"/>
```

Furthermore, explicating the temporality (or the life cycle) of classes can be automated using the temporal knowledge retrieved from CVS timestamps. These timestamps imply the life cycle of classes and properties and therefore timestamps can be used to automatically suggest a temporal enrichment of the ontology. But the temporal data got as a result from CVS timestamps have to be strictly segregated from other temporal intervals depicting validity periods of concepts in ontology revisions. For example, we might today get to know some new historical fact about some concept, say, Petsamo and hence in that case the time is modeled independently of the CVS timestamps.

Moreover, these kind of ontologies that define temporally indexed concepts can be used to annotate entities – like items of a museum collection – that are from some historical period. A complete life cycle for a concept can be generated by following a suitable method. Below is one suggestion for the method that is developed as a part of this thesis. The method consists of the following phases.

1. Retrieve all the different versions Ov_i of an ontology O. Also store the temporal intervals for each concept. Note that a concept life cycle can consist of several separate intervals.

2. Generate namespaces based on information about concept occurrences. For example, if concept c has occurred in versions 1.0, 1.1 and 1.2 then generate a namespace like "1.0-1.1-1.2" (if it does not exist already) and move concept c into it.

3. Create a new ontology where all classes from all ontology versions are depicted in their own namespaces.

4. Create a life cycle using the intervals, i.e., bind the classes with intervals created using the Time Ontology. Note that each interval has to be created once and reused for multiple classes having the same interval in their life cycle.

The Temporal sorting -method described previously in Chapter 3.3 has been implemented as an utility called TemporalSorter that uses Java [Sun04], interval checks from Allen Algebra and an implementation of the conditional probability calculation. The implementation is outlined in Algorithm 2.

input : Intervals T1 and T2
output: P $[T1 \mid T2]$=$\frac{T1 \cap T2}{T2}$
begin
  // Calculate the intersection of the given two temporal intervals ;
  if *T1.getBegin() < T2.getBegin()* **then**
    intersectBegin = T2.getBegin();
  end
  else
    intersectBegin = T1.getBegin();
  end
  if *T1.getEnd() < T2.getEnd()* **then**
    intersectEnd = T1.getEnd();
  end
  else
    intersectEnd = T2.getEnd();
  end
  T2length = $\frac{(T2.getEnd() - T2.getBegin())}{(1000*60*60*24)}$;
  intersectLength = $\frac{intersectEnd - intersectBegin}{(1000*60*60*24)}$;
end
// Return the conditional probability ;
return $\frac{intersectLength}{T2length}$;

Algorithm 2: Algorithm for calculating a weight as a conditional probability P $[T1 \mid T2]$.

The results depicting the ranked query results given by TemporalSorter can be seen in Table 4. The weights for each class can now be used e.g. to sort the query results for the user. Note that the implementation calculates the exact overlap. The months and days together with the differences between the duration of different years like leap-years have been taken into account.

The algorithm has been implemented as a Java [Sun04] implementation.

Rank	Class	Weight
1	Pechenga	0.56
2	Petsamo	0.44

Table 4: Ranked query results for *Petsamo* and *Pechenga* when *20th century* is selected.

4.4 Summary of Framework Tools

To enable and to aid transparent translating between ontology versions, the software and ontologies listed in Table 5 have been developed to form an ontology versioning framework. The framework aims to enable and to aid reasoning about a complete ontology version history.

Component	Description
Revision Ontology	An ontology defining metadata elements for ontology identification
Change-Bridging Ontology	A set of change bridges (mappings) as an ontology
ChangeTracker	A Java program utility for providing differences between two ontology versions and suggestions
Version Bridger	A Java program that tries a rule-based approach for semi-automatic forming of change bridges. This is a part of the future work.
Time Ontology	An ontology intended for representing life cycles, that is, validity periods of ontology concepts
TemporalSorter	A Java program for calculating and sorting query results using concept validity periods
TemporalInterval	A Java program that provides means for representing and comparing temporal intervals

Table 5: Components of the Ontology Versioning Framework.

Revision Ontology provides metadata elements for ontology identification. Elements include classes like author, language, publisher, title, date, priorVersion and id. To maximize the automation, many of the metadata elements are filled automatically by CVS interoperability.

Change-Bridging Ontology provides a set of change bridges (mappings) as classes

and properties for expressing the situation when a change has occurred. ChangeTracker checks the differences between two ontology versions and is able to use Version Bridger for providing suggestions about change bridges to be formed between ontology revisions. ChangeTracker uses Jena [CDD+03] for processing RDF and OWL. VersionBridger is a test utility to try a rule-based approach for semi-automatic forming of change bridges (mappings) between concepts from different ontology versions. Version Bridger is a part of the future work and therefore it is not described in detail in this thesis.

Time Ontology is intended for representing life cycles, that is, validity periods of ontology concepts including classes, properties, relations and instances. TemporalInterval provides means for representing temporal intervals and making comparisons (like meets, before, after, ...) between temporal intervals. TemporalSorter provides sorting of query results using the validity periods or other temporal intervals binded to concepts and in that uses services provided by TemporalInterval to compare intervals in the Allenian way. TemporalSorter implements conditional probability calculation and uses Java Calendar Libraries [Sun04] to provide exact values for sorting.

5 Conclusions

Classes, properties, individuals and relations between classes and individuals can be expressed as an ontology. As ontologies are altered to correct errors, to accommodate new information or to adjust the representation of the domain, they are said to evolve. Hence there has been a need for methods and means to manage the ontology evolution in order to ensure that applications using ontologies work properly. In this thesis ontology versioning has been studied by exploring previous research findings, gathering the requirements for ontology versioning, identifying the critical questions and finally presenting and implementing a framework for ontology versioning.

Previous research findings for ontology versioning were presented combining four important questions, namely identification, change-tracking, life cycle and transparent translating in the ontology evolution. These viewpoints were taken as a starting point and each area was discussed. Firstly, identification of the ontology can be done using an ontology collecting the revision metadata. The Revision Ontology combines the metadata elements – such as author, date, id, publisher – into a single package, which can be included in the edited ontology. Creating identifying metadata requires a lot of error-prone handwork and hence the metadata generation phase should be automated as much as possible. Although ontology versioning is still in its infancy — with no current ontology tools supporting versioning — good revising practices have been created in the area of software development. Concurrent Versions System, for example, provides the basic versioning automatically, including keeping a log of when, and why changes occurred and who conducted them. This can be utilized to fill up part of the Revision Ontology as was shown.

Change-tracking is another important requirement that was recognized. It is important to explicate changes, for example, in classes, instances, properties, relations or in a class hierarchy. It is also necessary to identify which change operations have produced the changes and further express the change as a mapping between evolved entities. In this thesis the idea of a Change Bridge was presented and defined to enable bridging the semantic gap between different versions of the ontology. Future work includes researching methods to automatically narrow down the set of possible change bridges between versions of an ontology.

Ontologies change as a function of time, meaning that time has a significant role in ontology evolution. Temporal knowledge about an ontology evolution is needed in order to answer questions like “When has the change occurred?” or “When have the

concepts been valid i.e. what are their life cycles?" Temporal overlap of concept life cycles can be used to sort the query results as was shown. Furthermore, using the Change Bridges in temporal sorting, complete life cycles of the evolved concepts can be taken into use and that way enable reasoning about a complete ontology version history.

References

All83 Allen, J. F., Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26,11(1983), pages 832–843.

Ber90 Berliner, B., CVS II: Parallelizing software development. *Proceedings of the Winter 1990 USENIX Conference*, USENIX Association, editor, Washington, DC, USA, January 22–26, 1990, USENIX, pages 341–352.

BG00 Brickley, D. and Guha, R. V., Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Wide Web Consortium, March 2000.

BLFM98 Berners-Lee, T., Fielding, R. and Masinter, L., RFC 2396: Uniform Resource Identifiers (URI): Generic syntax, Network Working Group, July 1998.

BM01 Biron, P. V. and Malhotra, A., XML Schema Part 2: Datatypes, W3C, 2001. http://www.w3.org/TR/xmlschema-2/, accessed February 10, 2004.

BvHH+04 Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. and Stein, L. A., OWL web ontology language reference. Recommendation, World Wide Web Consortium, February 2004. http://www.w3.org/TR/owl-ref/, accessed February 11, 2004.

CDD+03 Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. and Wilkinson, K., Jena: Implementing the semantic web recommendations. Technical Report HPL-2003-146, HP Labs, December 24, 2003.

Doe03 Doerr, M., The CIDOC conceptual reference module: an ontological approach to semantic interoperability of metadata. *AI Magazine*, 24,3(2003), pages 75–92.

Dom98 Domingue, J., Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the Web. *Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling and Management, Banff, Canada*, 1998.

DWM01 Das, A., Wu, W. and McGuinness, D., Industrial strength ontology management. *Proceedings of the First Semantic Web Working Symposium (SWWS'01)*, Stanford University, California, USA, July 2001, pages 17–38.

Fen00 Fensel, D., *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2000.

FFR97 Farquhar, A., Fikes, R. and Rice, J., The ontolingua server: a tool for collaborative ontology construction. *International Journal of Human-Computer Studies, 46, 707-727, 1997.*

For00 Force, A. I. T., *Categories for the Description of Works of Art (CDWA)*. The Getty Information Institute, Los Angeles, California, 2000.

GEF+99 Grosso, W. E., Eriksson, H., Fergerson, R. W., Gennari, J. H., Tu, S. W. and Musen, M. A., Knowledge modeling at the millennium (the design and evolution of Protégé-2000). *Twelfth Banff Workshop on Knowledge Acquisition, Modeling, and Management. Banff, Alberta, 1999.*

Gru93 Gruber, T. R., A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5, 1993, pages 199–220.

Hef01 Heflin, J., Towards the semantic web: Knowledge representation in a dynamic, distributed environment. PhD Thesis, University of Maryland, 2001.

HFLW02 Hendler, J., Fensel, D., Liebermann, H. and Wahlster, W. *Spinning the Semantic Web*. MIT Press, 2002.

HH00 Heflin, J. and Hendler, J., Dynamic ontologies on the web. *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*. AAAI/MIT Press, Menlo Park, CA, 2000, pages 443–449.

HHL99 Heflin, J., Hendler, J. and Luke, S., Coping with changing ontologies in a distributed environment. *Ontology Management. Papers from the AAAI Workshop. WS-99-13*. AAAI Press, 1999, pages 74–79.

ISO88 ISO (International Organization for Standardization), ISO 8601 Representations of dates and times, 1988.

KDFO02 Klein, M., Ding, Y., Fensel, D. and Omelayenko, B., Ontology management — storing, aligning and maintaining ontologies. In *Towards The Semantic Web: Ontology-Driven Knowledge Management*, Davies, J., Fensel, D. and van Harmelen, F., editors, John Wiley & Sons, 2002.

KF01 Klein, M. and Fensel, D., Ontology versioning on the Semantic Web. *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA, July 30 – August 1, 2001, pages 75–91.

KFKO02 Klein, M., Fensel, D., Kiryakov, A. and Ognyanov, D., Ontology versioning and change detection on the web. *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, number 2473 in LNCS, Sigüenza, Spain, October 1–4, 2002.

Kle01 Klein, M., Combining and relating ontologies: an analysis of problems and solutions. *Workshop on Ontologies and Information Sharing, IJCAI'01*, Gomez-Perez, A., Gruninger, M., Stuckenschmidt, H. and Uschold, M., editors, Seattle, USA, August 4–5, 2001.

Kle02 Klein, M., IST Project 2001-33052 WonderWeb: Ontology infrastructure for the semantic web. Versioning of distributed ontologies. Technical Report, Department of Computer Science, Vrije Universiteit Amsterdam, December 2002.

Kor04 Korpilahti, T., Architecture for distributed development of an ontology library. Master's thesis, Helsinki University of Technology, 2004.

LS99 Lassila, O. and Swick, R. R., Resource Description Framework (RDF): Model and Syntax Specification. Recommendation, World Wide Web Consortium, February 1999.

ME93 M. Eisinger, H. O., Calculus: resolution. In *Handbook of Logic in Artificial Intelligence and Logic Programming - Vol 1: Logical Foundations. Oxford, Clarendon Press*, Gabbay, D. M., Hogger, C. J. and Robinson, J. A., editors, 1993, page 211.

MFR+00 McGuinness, D. L., Fikes, R., Rice, J., and Wilder, S., The Chimaera ontology environment. *Proceedings of the The Seventeenth National Conference on Artificial Intelligence, Austin, Texas, July 2000.*

MWK00 Mitra, P., Wiederhold, G. and Kersten, M., A graph-oriented model for articulation of ontology interdependencies. *Extending DataBase Technologies, EDBT 2000, Konstanz, Germany, 2000.*

NK03 Noy, N. and Klein, M., Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems 5, 2003.*

NM99 Noy, N. F. and Musen, M. A., Smart: Automated support for ontology merging and alignment. *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management, Banff, Canada*, July 1999.

NM01 Noy, N. F. and McGuinness, D. L., Ontology development 101: A guide to creating your first ontology. Internal note, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

NM03 Noy, N. F. and Musen, M. A., Ontology versioning as an element of an ontology-management framework. IEEE Intelligent Systems, 2003.

NP03 Niles, I. and Pease, A., SUMO (Suggested Upper Merged Ontology), Temporal Concepts. Palo Alto, California, 2003, Teknowledge. http://ontology.teknowledge.com/, accessed December 19th, 2003.

OWL03 OWL Web Ontology Language Reference, W3C, December 2003. www.w3.org/TR/owl-ref/, accessed January 10th, 2004.

PH04 Pan, F. and Hobbs, J. R., Time in OWL-S. *Semantic Web Services.* AAAI Spring Symposium Series, 2004.

Sau98 Saur, M., IFLA Study Group, Functional requirements for bibliographic records(FRBR), Final report. *International Federation of Library Associations and Institutions, UBCIM Publications - New Series Vol 19*, 1998.

SMMS02 Stojanovic, L., Maedche, A., Motik, B. and Stojanovic, N., User-driven ontology evolution management. *Proceedings of the 13th International Conference EKAW 2002*, Sigüenza, Spain, 2002, EKAW, pages 285–300.

Sun04 Sun Microsystems, Java specification, 2004. http://java.sun.com, accessed February 10, 2004.

SWM04 Smith, M. K., Welty, C. and McGuinness, D. L., W3C OWL Web Ontology Language Guide W3C Recommendation, February 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/, accessed February 11, 2004.

vdB95 van den Berg, J., Subject retrieval in pictorial information systems. *In Proceedings of the 18th International Congress of Historical Sciences, Montreal, Canada*, 1995.

VJBCS97 Visser, P. R. S., Jones, D. M., Bench-Capon, T. J. M. and Shave, M. J. R., An analysis of ontological mismatches: Heterogeneity versus interoperability. *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.

Woo00 Wooldridge, M., *Reasoning About Rational Agents*. MIT Press, 2000.

YSA03 Interview of Eeva Kärki (Helsinki University Library in Finland) on the development of General Thesaurus in Finnish (Yleinen Asiasanasto, YSA), September 30th, 2003.