D3.1.1.b State-of-the-Art on Ontology Evolution

Peter Haase and York Sure
(Institute AIFB, University of Karlsruhe)

Abstract.
EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D3.1.1.b (WP3.1)
This document is an informal deliverable provided to SEKT partners. The main aim of this document is to provide an overview of existing research results and ongoing research in the are of ontology evolution. It further identifies relevant open research questions with respect to ontology evolution that will be addressed within the SEKT project.
Keyword list: Ontology Management, Ontology Evolution, Ontology Versioning

Document Id.	SEKT/2004/D3.1.1.b/v0.5
Project	SEKT EU-IST-2003-506826
Date	August 12th, 2004
Distribution	informal deliverable, project internal

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contactperson: John Davies
E-mail: john.nj.davies@bt.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contactperson: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1891, Fax: +44 114 222 1810
Contactperson: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006 Madrid
Spain
Tel: +34 913 349 797, Fax: +49 34 913 349 799
Contactperson: Richard Benjamins
E-mail: rbenjamins@isoco.com

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912, Fax: +49 721 50980911
Contactperson: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contactperson: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Empolis GmbH
Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540, Fax: +49 631 303 5507
Contactperson: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592, Fax: +49 721 608 6580
Contactperson: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Innsbruck
Institute of Computer Science
Techikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475, Fax: +43 512 507 9872
Contactperson: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Kea-pro GmbH
Tal
6464 Springen
Switzerland
Tel: +41 41 879 00, Fax: 41 41 879 00 13
Contactperson: Tom Bösser
E-mail: tb@keapro.net

Sirma AI EOOD (Ltd.)
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768, Fax: +359 2 9768 311
Contactperson: Atanas Kiryakov
E-mail: naso@sirma.bg

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vallès)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contactperson: Pompeu Casanovas Romeu
E-mail: pompeu.casanovasquab.es

Contents

Chapter 1

Introduction

1.1 The SEKT Big Picture

This report is part of the work performed in workpackage (WP) 3 on "Ontology and Metadata Management". As shown in Figure 1.1 this work belongs to the central part of the research and development WPs in SEKT. Quite naturally it is closely connected with Ontology Generation and Metadata Generation, in particular we will integrate parts of their technologies. We are focussing on how to manage ontologies (and related metadata) and their evolution over time. We will provide as part of WP3.1 a basic infrastructure for ontology management. We will extend this in WP3.2 and WP3.3 with functionalities for data-driven change discovery and usage tracking, i.e. with means to adapt ontologies according to underlying domain knowledge in form of documents on the one hand and the usage of ontologies in applications by users on the other hand. As part of WP7 Methodology we will closely collaborate with the case study partners to apply our technologies within the case studies (see e.g. [EGH+04b, EGH+04a, ST04] and following ones).

1.2 Ontologies

Initially introduced by Aristotle, ontologies recently have become a topic of interest in computer science. Ontologies provide a shared understanding of a domain of interest to support communication among human and computer agents, typically being represented in a machine-processable representation language. Thus, ontologies are seen as key enablers for the Semantic Web [BLHL01]. Standards for ontology languages include the layered W3C standards XML/S, RDF/S and OWL. There exist numerous scientific and commercial tools for creating and maintaining ontologies (see Chapter 3), which have been used to build applications based on ontologies, including the areas of knowledge management, engineering disciplines, medicine or bio-informatics.

However, those pieces of knowledge so far have been treated mainly as being static.

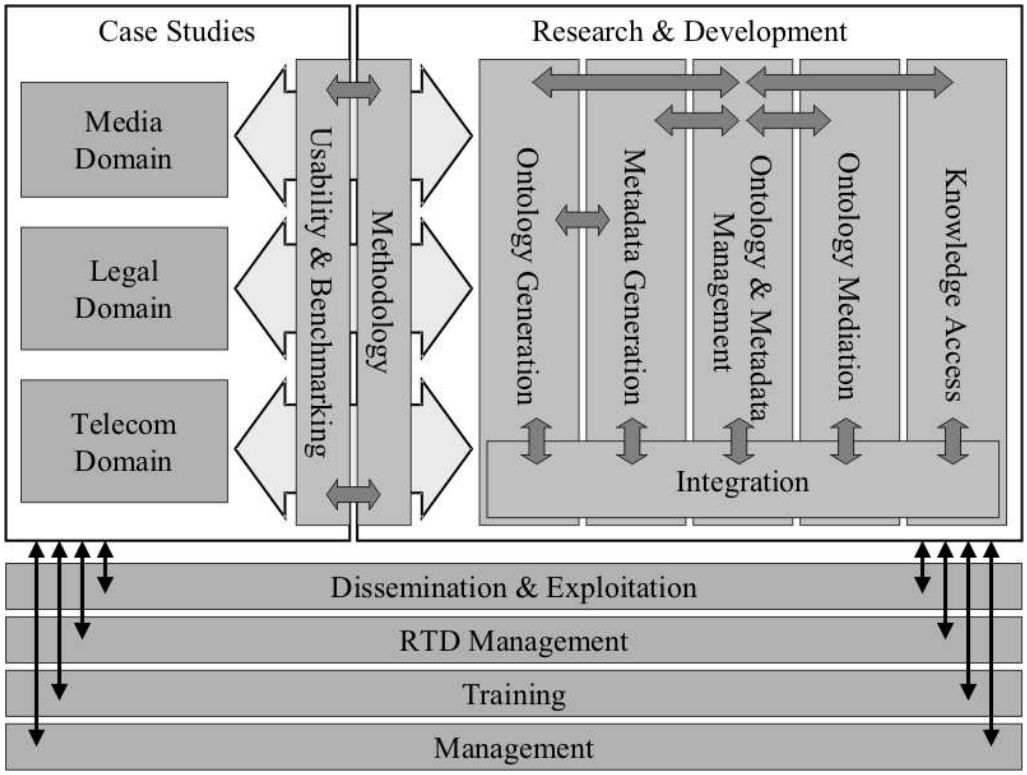

Figure 1.1: The SEKT Big Picture

In reality they evolve over time, sometimes they even have a highly dynamic nature (see e.g. Peer-to-Peer scenarios such as Bibster [HEHS04]). Domain changes, adaptations to different tasks, or changes in the conceptualization require modifications of the ontology.

1.3 Definition

We will now define some terms which are most relevant for this document.

According to [Sto04], "Ontology Evolution is the timely adaptation of an ontology to the arisen changes and the consistent propagation of these changes to dependent artefacts." The author describes ontology evolution as a process, as changes in the ontology can cause inconsistencies in other parts of the ontology, as well as in the dependent artefacts, the . The ontology evolution process encompasses the set of activities, both technical and managerial, that ensures that the ontology continues to meet organizational objectives and users needs in an efficient and effective way.

In [SMMS02] the authors identify a possible six-phase evolution process, the phases being: (1) change capturing, (2) change representation, (3) semantics of change, (4) change implementation, (5) change propagation, and (6) change validation. In the fol-

lowing, we will use this evolution process as the basis for an analysis of state-of-the-art technology.

Further, it is important to distinguish between the management, modification, evolution and versioning of ontologies. In this document we follow the terminology of [Sto04], which has been adapted from the terminology from the database community [Rod95]:

- **Ontology management** is the whole set of methods and techniques that is necessary to efficiently use multiple variants of ontologies from possibly different sources for different tasks. Therefore, an ontology management system should be a framework for creating, modifying, versioning, querying, and storing ontologies. It should allow an application to work with an ontology without worrying about how the ontology is stored and accessed, how queries are processed, etc.;
- **Ontology modification** is accommodated when an ontology management system allows changes to the ontology that is in use, without considering the consistency;
- **Ontology evolution** is accommodated when an ontology management system facilitates the modification of an ontology by preserving its consistency;
- **Ontology versioning** is accommodated when an ontology management system allows handling of ontology changes by creating and managing different versions of it.

1.4 Outline

The remainder of this document is structured as follows. In Chapter 2 we present an overview of the relevant research area. In Chapter 3 the most relevant existing tools for ontology evolution and versioning are being described. Chapter 4 presents related research according to the evolution process described in Section 2.1, but also covers ontology versioning and evolution/versioning in database systems. Finally, we conclude in Chapter 5.

Chapter 2

Overview of the area

2.1 Ontology Evolution Process and Frameworks

In [SMMS02] the authors identify a possible sixphase evolution process, the phases being: (1) change capturing, (2) change representation, (3) semantics of change, (4) change implementation, (5) change propagation, and (6) change validation. In the following, we will use this evolution process as the basis for an analysis of state-of-the-art technology.

Change Capturing The process of ontology evolution starts with capturing changes either from explicit requirements or from the result of **change discovery** methods, which induce changes from existing data. Explicit requirements are generated, for example, by ontology engineers who want to adapt the ontology to new requirements or by the end-users who provide the explicit feedback about the usability of ontology entities. The changes resulting from this kind of requirements are called *top-down* changes. Implicit requirements leading to so-called *bottom-up* changes are reflected in the behaviour of the system and can be discovered only through the analysis of this behaviour. [Sto04] defines three types of change discovery: structure-driven, usage-driven and data-driven. Whereas structure-driven changes can be deduced from the ontology structure itself, usage-driven changes result from the usage patterns created over a period time. Data-driven changes are generated by modifications to the underlying dataset, such as text documents or a database, representing the knowledge modeled by an ontology.

Change Representation To resolve changes, they have to be identified and represented in a suitable format. That means, the change representation needs to be defined for a given ontology model. Changed can be represented on various levels of granularity, e.g. as elementary or complex changes. A common practice is to provide a taxonomy or ontology of changes for a given ontology model.

Semantics of Change The semantics of change refers to the effects of the change on the ontology itself, and, in particular the checking and maintenance of the ontology consistency after the change application. The meaning of consistency very much depends on the underlying ontology model. It can for example be defined using a set of constraints, as in the KAON ontology model, or it can be given a model-theoretic definition.

Change Propagation Ontologies often reuse and extend other ontologies. Therefore, an ontology update might also corrupt ontologies depending on the modified ontology (through the inclusion, mapping integration, etc.) and consequently, all the artefacts based on these ontologies. The task of the change propagation phase of the ontology evolution process is to ensure consistency of *dependent artefacts* after an ontology update has been performed. These artefacts may include dependent ontologies, instances, as well as application programs running against the ontology.

Change Implementation The role of the change implementation phase of the ontology evolution process is (i) to inform an ontology engineer about all consequences of a change request, (ii) to apply all the (required and derived) changes and (iii) to keep track about performed changes.

Change Validation There are numerous circumstances where it may be desired to reverse the effects of the ontology evolution, to name just a few:

- The ontology engineer may fail to understand the actual effect of the change and approve the change that should not be performed;
- It may be desired to change the ontology for experimental purposes;
- When working on an ontology collaboratively, different ontology engineers may have different ideas about how the ontology should be changed.

It is the task of the change validation phase to recover from these situations. Change validation enables justification of performed changes and undoing them at user's request. Consequently, the usability of the ontology evolution system is increased.

2.2 Ontology Versioning

Ontology versioning is a stronger variant of handling changes to ontologies: While ontology evolution is concerned about the ability to change an ontology without losing data and by maintaining consistency, ontology versioning allows to access the data through different variants of the ontology. In addition to managing the individual variants of the ontology themselves, it is also important to manage the derivation relations between the

variants. These derivation relations then allow to define the notions of compatibility between versions, mapping relations between versions, as well as transformations of data corresponding to the various versions.

2.3 Evolution and Versioning in Database Systems

The problem of schema evolution has been extensively studied especially in the context of object-oriented databases. Dynamic schema evolution in databases is defined as managing schema changes in a timely manner without loss of existing data while the database system continues to be operational and without significantly impacting day-to-day operations of the database. Particular problems addressed are cascading changes (changes required to other parts of the schema as a result of a change), ensuring consistency of the schema, and propagation of the changes to the corresponding database.

Although there are significant differences between schema evolution and ontology evolution [NK03], many of the methods and technologies developed for schema evolution can be applied or adapted to ontology evolution.

2.4 Evolution and Versioning for Other Paradigms

The problem of evolution and versioning is also present in other application areas of information systems.

For example, Concurrent Versions Systems (CVS) allow the concurrent update to files while maintaining the version history of those files as well as detecting and resolving conflicts in updates to the files.

Another application are is the maintenance of knowledge-base systems and belief revision, where a knowledge base (e.g. the believes of an agent) needs to be update to incorporate new information while maintaining consistency of the knowledge base.

Chapter 3

Existing Tools

There are only few existing tools that support the complete ontology evolution process. We therefore also provide an overview of tools that only support specific aspects in the ontology evolution process. Further, we also cover non-commercial tools, i.e. tools that have come out of research projects. Before we illustrate the tools which already exist for ontology evolution and versioning we begin with one of the most prominent systems for versioning, the CVS, and discuss its drawbacks for the usage with ontologies.

3.1 Concurrent Version System – CVS

The very popular concurrent version system (CVS[1]) initially was a collection of scripts to simplify the handling of the revision control system (RCS). RCS operates in a file-centric way by using a "lock-modify-unlock"-style. CVS still relies on the RCS file format for storing versioning information, but it extends RCS e.g. by supporting network capabilities, by separating local copies and central repositories and by allowing parallel access of multiple users. The CVS basic version control functionality maintains a history of all changes made to directory trees, *i.e.* a hierarchy of file folders which might contain arbitrary file formats (often text files). The complete functionalities are described in the "official" manual for CVS [C+03].

However, CVS works on the syntactical level, not on the conceptual. *I.e.*, it is not capable of versioning objects and in particular not capable of versioning ontological entities and their complex structure. The underlying `diff` operation is capable of showing the syntactical differences between two files (based on the differences of text lines). Therefore, it is suitable to act as a very primitive versioning system *e.g.* for RDF/S or OWL *files*.

In a nutshell, standard `diff`, and thus CVS, compares file versions at line-level or

[1] Available freely for download at `http://www.cvshome.org/`

at character-level, highlighting the lines that textually differ in two versions. Actually needed is a comparison of ontologies at a structural level, showing which definitions of ontological concepts or relationships have changed.

Nevertheless, CVS can has already been used for inspiration for the ontology versioning systems such as OntoView (see [Kle04] and next Section 3.4).

3.2 Ontology Editors

Ontology editors are tools that allow users to visually manipulate ontologies. In this section, we evaluate ontology editors in terms of the requirements for the ontology evolution. We select three ontology editors that are most frequently used in the Semantic Web community, a more complete overview can be found in [GPAFL+02].

- Protégé[2] (cf. [NFM00]) is a graphical and interactive ontology-design and knowledge-acquisition environment that is being developed by the Stanford Medical Informatics group (SMI) at Stanford University. Its knowledge model is compatible with OKBC (cf. [CFF+98]). Its component-based architecture enables system builders to add new functionality by creating appropriate plug-ins. The Protégé-OWL plugin extends the Protégé platform into an ontology editor for the OWL;
- OntoEdit[3] (cf. [SAS03, SEA+02]) is an ontology engineering environment supporting the development and maintenance of ontologies by graphical means. OntoEdit is built on top of a powerful internal ontology model. This paradigm supports representation-language neutral modelling as much as possible for concepts, relations and axioms. Several graphical views onto the structures contained in the ontology support modelling the different phases of the ontology engineering cycle. It has an interface to Ontobroker, an F-Logic Inference Engine;
- OilED[4] (cf. [BHGS01]) has been developed by the University of Manchester. It is a simple freeware ontology editor, which allows the user to build ontologies using OIL and OWL, and it is not intended as a full ontology development environment. Consistency checking and automatic classification of the ontologies written with it can be performed using the FaCT reasoner.

All editors allow modifications to ontologies in terms of elementary ontology changes. Even though composite changes allow an ontology engineer to update an ontology without having to find the right sequence of elementary modifications, most of the existing ontology editors do not include composite changes. Only OntoEdit provides support for some composite changes (e.g. copy).

[2] `http://protege.stanford.edu/`
[3] `http://www.ontoprise.de/com/co_produ_tool3.htm`
[4] `http://oiled.man.ac.uk`

Most of the existing systems for the ontology development provide only one possibility for realising a change, and this is usually the simplest one. For example, the deletion of a concept always causes the deletion of all its subconcepts. It means that users are not able to control the way the changes are performed. Consequently, customisation is not supported at all.

Moreover, the users do not obtain explanations why a particular change is necessary (transparency). In OntoEdit, the user only obtains the information about numbers of induced changes but without providing more details. None of existing editors warns ontology engineers about changes in the included ontologies.

Furthermore, there is no possibility to undo effects of changes (reversibility). Both, Protégé and OntoEdit, have an Edit menu with the Undo/Redo options. However, the performed changes are kept in the memory so that they are lost when the ontology/editor is closed.

Regarding auditing and the logging of changes, OilEd provides an activity log. However, it records connections to the reasoner, not all ontology modifications. Protégé also has the command history option but in the version we were dealing with it was useless since it was disabled.

3.3 Ontology Evolution in KAON

KAON is an open-source ontology management infrastructure targeted for semantics-driven business applications. It provides a comprehensive implementation allowing easy ontology management and application. Important focus of KAON is on integrating traditional technologies for ontology management and application with those used typically in business applications. A more detailed technical description of the KAON components can be found in [GSV04].

Roughly, KAON components can be divided into three layers:

- Applications and Services Layer realizes UI applications and provides interfaces to non-human agents. Among many applications realized, OntoMat-SOEP provides ontology and metadata engineering capabilities. It realizes many requirements related to ontology evolution and is described next in more detail.
- KAON API as part of the Middleware Layer is the focal point of KAON architecture since it realizes the model6 of ontology based applications. The bulk of requirements related to ontology evolution is realized in this layer and is described in the next section.
- Data and Remote Services Layer provides data storage facilities. This layer also realizes concurrency and transactional atomicity of updates. Further elaboration of this layer is out of scope for this paper.

The focal point of the KAON architecture is its ontology API (KAON API), consisting of a set of interfaces for access to ontology entities. For example, there are interfaces Concept, Property and Instance, which contain methods for accessing ontology concepts, properties and instances, respectively.

The API incorporates important elements required for ontology management and evolution:

- Evolution logging is responsible for keeping track of the ontology changes in an evolution log in order to be able to reverse them at the user s request. Further, the evolution log is also used by the distributed ontology evolution;
- Change reversibility enables undoing and redoing changes made in an ontology. Consequently, changes can be executed in reverse order thus forcing the ontology to return to the conditions prior to the change execution;
- Evolution strategy is responsible for ensuring that all changes applied to the ontology leave the ontology in a consistent state and for preventing illegal changes. Also, the evolution strategy allows the user to customise the evolution process;
- Evolution graph enables ontology engineers to enhance a set of changes with their own changes and to resolve them;
- Ontology inclusion facilities, together with the dependent evolution, are responsible for managing multiple ontologies within one node; ” Ontology replication facilities, together with the distributed evolution, are responsible for enabling the reuse and the management of distributed ontologies;
- Change discovery includes the means for the discovery of problems in an ontology and for making recommendation for their resolution;
- Usage logging is responsible for keeping track of the end-users interactions with ontology-based applications in order to adapt ontologies to the users needs.

3.3.1 Ontology Evolution in KAON API

Before the ontology evolution process is started, a particular evolution strategy must be configured. Changes to the ontology are performed by assembling elementary and composite changes into a sequence. However, before the ontology is actually updated, this sequence is passed to the present evolution strategy in the semantics of change phase, resulting in an extended sequence of changes. To ensure atomicity of updates, either all or no change from the extended sequence of changes should succeed, so validity of change sequence is checked before any updates are actually performed. Transparency is realized by presenting the extended sequence of changed to the user for approval. To further aid the understanding of why some changes are performed, the evolution strategy may group

related elementary actions and provide explanations why particular change is necessary, thus greatly increasing the chances that all side-effects of changes will be properly understood. After changes are reviewed by the user, they are passed to the ontology and executed, performing steps from the change implementation phase. It is obvious that for each elementary change there is exactly one inverse change that, when applied, reverses the effect of the original change. With such infrastructure in place, it is not hard to realize the reversibility: to reverse the effect of some extended sequence of changes, a new sequence of inverse changes in reverse order needs to be created and applied. An evolution log associates additional information with each change. Effectively, the log is treated as an instance of a special evolution ontology (cf. Section 4.1.4) consisting of concepts for each change, making it is easy to add meta-information to log entries. Structure of the log may be easily customized by editing the evolution ontology. Further, available services for persisting ontology data may be used to persist the log, removing the need to devise yet another type of persistent storage. Evolution logging and reversibility services are provided as special services of KAON API, allowing different applications reuse these powerful features. E.g., actions performed in one application may be easily reverted in another.

3.3.2 Ontology Evolution in the OI-modeller

As mentioned in the previous section, the ontology evolution is primarily realised through the KAON API. However, UI applications provide human-computer interaction for the evolution, whose primary role is to present the change information in an orderly way, allowing easy spotting of potential problems. Also, any application that changes the ontology must realise the reversibility requirement in its user interface as well. Part of the KAON framework is the OI-modeller, an ontology and metadata engineering tool. It is an end-user application that realises a graph-based user interface for single, dependent and distributed ontology development. OI-modeller supports ontology evolution at the user level.

Currently evolution requirements are realised within the OI-modeller, as follows:

- An ontology engineer may set up the desired evolution strategy. It can be seen that an evolution strategy consisting of several resolution points. For each resolution point the ontology engineer must choose appropriate elementary evolution strategy;
- Before changes are performed, the system computes the set of additional changes that must be applied. The impact of a change is reported to the ontology engineer. Presentation of changes follows the progressive disclosure principle: related changes are grouped together and organised in a tree-like form. The ontology engineer initially sees only the general description of changes. If she is interested in details, she can expand the tree and view complete information. Only when the ontology engineer agrees will the changes be applied to the ontology. The ontology engineer may cancel the operation before it is actually performed.

- An unlimited undo-redo function is provided. Although this function is by large the responsibility of the KAON API, the user interface is responsible for restoring the visual context after an undo operation. For example, if a concept in hierarchy was selected and then deleted, when operation is undone, the same concept must be selected. If the hierarchy was scrolled in the meanwhile, the original scroll position must be restored. These features are necessary for the ontology engineer to quickly recognise a familiar state and proceed with her work. If not done properly, although an action is undone, the ontology engineer may not realise this and may mistakenly request another undo operation.

A detailed description about the OI-modeler and the KAON API including the support for evolution can be found in [GSV04].

3.4 OntoView

In [KFKO02] the authors describe the design of a web-based system that helps users to manage changes in ontologies. The system helps to keep different versions of web-based ontologies interoperable, by maintaining not only the transformations between ontologies, but also the conceptual relation between concepts in different versions. OntoView is inspired by the Concurrent Versioning System CVS (see previous section), which is used in software development to allow collaborative development of source code. The first implementation is also based on CVS and its web-interface CVSWeb.

The versioning system of OntoView provides the following functionalities:

1. Reading changes and ontologies
2. Identification of ontologies
3. Comparing ontologies at a conceptual level
4. Analyzing effects of changes, e.g. by checking consistency
5. Exporting changes

3.5 OntoManager

OntoManager[SHG03] is a tool for guiding ontology managers through the modification of an ontology with respect to users' needs. It is based on the analysis of end-users' interactions with the ontology-based applications, which are tracked in a usage-log.

OntoManager has been designed to provide the methods and tools that support the ontology managers in managing and optimising the ontology according to the users' needs.

The system incorporates mechanisms that assess how the ontology (and by extension the application) is performing based on different criteria, and then enable to take action to optimise it. One of the key tasks is to check how the ontology fulfils the perceived needs of the users. In that way, an in-depth view of the users' perspective on the ontology and the ontology-based application is obtained, since on the top of this ontology the application is going to be conducted. The technique that can be used to evaluate/estimate the user needs depends on the information source. By tracking user interactions with the application in a log file, it is possible to collect useful information that can be used to assess what the main interests of the users are. In this way, it is avoided to ask the users explicitly, since they tend to be reluctant to provide the feedback via filling questionnaires or forms.

Conceptually, the OntoManager consists of three modules:

- The Data Integration Module that aggregates, transforms and correlates the usage data;
- The Visualisation Module that makes the integrated usage data more useful for human beings by presenting the data in a comprehensible visual form;
- The Analysis Module that provides guidance for adapting and improving the ontology with respect to the users' needs.

Withe respect to ontology evolution, the "Analysis Module" is most important: In particularly, there are two task of the Analysis module. In particularly, there are two task of the Analysis module:

1. Ontology Evolution that provides guidance in the process of modifying the ontology and ensure the consistency of the updated ontology. This module keeps track of the changes and has the possibility to undo any action taken upon the ontology. The OntoManager imports the functionalities related to the ontology evolution process that are elaborated in [SMMS02].
2. Crawling that completes newly created concepts with the most promising instances that can be found in an intranet or internet.

3.6 TextToOnto

TextToOnto [MV01] is a tool suite built upon KAON [KK04] in order to support the ontology engineering process by text mining techniques. Providing a collection of independent tools for both automatic and semi-automatic ontology extraction it assists the user in creating and extending OIModels. Moreover, efficient support for ontology maintenance is given by modules for ontology pruning and comparison. Further information can be found, e.g., in [GSV04].

Since TextToOnto does not keep any references between the ontology and the text documents it has been extracted from, it does not allow for mapping textual changes to the ontology. Therefore data-driven change discovery is not (yet) supported by current versions of TextToOnto. We will focus in the follow-up Text2Onto[5], a complete re-implementation and the official successor of TextToOnto, on overcoming this deficiency. A detailed requirements analysis, together will a first architecture proposal, will be released as part of Task 3.3.

[5] `http://ontoware.org/projects/text2onto/`

Chapter 4

Past and current research

4.1 Ontology Evolution Process and Frameworks

We again use the evolution process presented in [SMMS02] to structure the structure this chapter. However, we also would like to mention other approaches to the evolution process and frameworks.

In [PK97] the authors define three major steps in the schema evolution process: 1) request specification, 2) identification of changes, and 3) implementation. The change capturing phase and the change validation phase are not covered. Regarding the core evolution process dealing with the consistency of a schema and its dependent artefacts, it does not treat the semantics of change problem and requires writing the transformations to update data if they do not have to be lost. Regarding the implementation of changes it allows to realise the evolution by view, by version or by the immediate update.

[KN03] introduces a component-based framework for ontology evolution. It is based on the different representations of ontology changes. The approach proposes a framework that integrates these representations. It covers the following tasks: (i) data transformation; (ii) ontology update; (iii) consistent reasoning; (iv) verification and approval; and (v) data access. The last task is related to ontology versioning.

4.1.1 Change Capturing

Please note that we will refine this section (including the following subsections) in future as part of our work in tasks T3.2 and T3.3.

Usage Driven Change Discovery

Once ontologies reach certain levels of size and complexity, the decision about which parts are further relevant and which are outdated is a huge task for ontology engineers. Usage patterns of ontologies and their metadata allow for a detection of often or less often used parts, thus reflecting e.g. the interests of users in parts of ontologies. They can be e.g. derived from tracking querying and browsing behaviors of users during the application of ontologies.

Data Driven Change Discovery

An ontology is often learnt or constructed in order to reflect the knowledge more the less implicitly given by a number of documents or a database. Therefore, any change to the underlying data set such as a newly added document or a changed database entry might require an update of the ontology. Data-driven Change Discovery can be defined as the task of deriving ontology changes from modifications to the knowledge representation it has been constructed from. Or, more formally:

Definition: Let *D* be a data set containing explicit or implicit knowledge, which is modified by a sequence *C1,...Cn* of change operations. And let the knowledge in D be explicitly represented by an ontology *O*. Then Data-driven Change Discovery can be defined as the task of adapting *O* in order to reflect the changes *C1,...Cn*.

A slightly different definition is given by [Sto04], who defines Data-driven Change Discovery as the problem of deriving ontological changes from the ontology instances by applying techniques such as data-mining, Formal Concept Analysis (FCA, [GW99]) or various heuristics. For example, one possible heuristic might be: If no instance of a concept C uses any of the properties defined for C, but only properties inherited from the parent concept, C is not necessary. An implementation of this notion of Data-driven Change Discovery is included in the KAON tool suite [KK04, GSV04].

One very obvious difference between these two definitions is, that the latter always assumes an existing ontology, while the former can be applied to an empty ontology as well, but requires an evolving data set associated with this ontology. Moreover the following prerequisites must be fulfilled:

1. Knowledge about *general* relationships between data and ontology is required, since in case of newly added or modified data, additional knowledge has to be extracted and represented by the ontology.

2. Knowledge about *concrete* relationships between the data and ontology concepts, instances and relations is needed, because deleting or modifying information in the data set might have an impact on existing entities in the ontology.

If the ontology creation process is done manually, for example by a knowledge engineer, then both kinds of knowledge are represented somewhere in the mind of this knowledge engineer. In that case Data-driven Change Discovery also has to be done manually. On the other hand, if the process of creating the ontology is done semi- or fully automatically with the help of an ontology learning system such as TextToOnto [MV01], this knowledge has to be represented explicitly by the system. Of course, the first kind of knowledge is always given by the concrete implementation of ontology learning algorithms which are used. Therefore, in order to enable an existing ontology learning system to support Data-driven Change Discovery, it is necessary to make it store all available knowledge about concrete relationships between ontology entities and the data set. A more detailed requirements analysis will included in deliverable [ref], due to [?].

4.1.2 Change Representation

[Sto04] derives a set of ontology changes for the KAON ontology model. The author specifies fine-grained changes that can be performed in the course of the ontology evolution. They are called elementary changes, since they cannot be decomposed into simpler changes. A taxonomy of elementary changes is derived as the cross product of the set of entities of the ontology model and the meta-change transformations *add* and *remove*.

The author also mentions that this level of change representation is not always appropriate and therefore introduces the notion of composite changes: A composite change is an ontology change that modifies (creates, removes or changes) one and only one level of neighborhood of entities in the ontology. Examples for these composite changes would be: “Pull concept up”, “Concept Copy”, “Split Concept”, etc. “ Further, the author introduces complex changes: A complex change is an ontology change that can be decomposed into any mix of at least two elementary and composite ontology changes.

In [KN03] Klein and Noy also state that information about change can be represented in many different ways. They describe different representations and propose a framework that integrates them. They show how different representations in the framework are related by describing some techniques and heuristics that supplement information in one representation with information from other representations. They further present an ontology of change operations, which is the kernel of the framework.

[Kle04] describes a set of changes for the OWL ontology language, based on an OWL meta-model. Unlike the previously mentioned set of KAON ontology changes, the author considers also *Modify*-operations in addition to *Delete* and *Add*-operations. Further, for the taxonomy contains *Set* and *Unset*-operations for properties (e.g. to set transitivity). The author introduces an extensive terminology of change operations along two dimensions: *atomic* vs. *composite* and *simple* vs. *rich*:

	simple	rich
atomic	basic	complex
composite	complex	complex

Atomic operations are operations that cannot be subdivided into smaller operations, whereas *composite operations* provide a mechanism for grouping operations that constitute a logical entity. *Simple changes* can be detected by analyzing the structure of the ontology only, whereas *rich changes* incorporate information about the implication of the operation on the logical model of the ontology, for their identification one thus needs to query the logical theory of the ontology (e.g. *ModifyDomainToSuperclass*). The authors also proposes a method for finding complex ontology changes. It is based on a set of rules and heuristics to generate a complex change from a set of basic changes.

As one can easily see, the terminology of [Kle04] is not consistent with the terminology introduced in [Sto04]. Simply speaking, *elementary* and *complex* changes in [Sto04] correspond to *atomic* and *composite* changes in [Kle04], respectively. In [Sto04], there is no explicit corresponding distinction for *simple* vs. *rich* changes.

Both [Sto04] and [Kle04] present an "ontology for ontology changes" for their respective ontology language and identified change operations.

There exist models for change representations for other ontology languages: A formal method for tracking changes in the RDF repository is proposed in [OK02]. The RDF statements are pieces of knowledge they operate on. The authors argue that during the ontology evolution, the RDF statements can be only deleted or added, but not changed. Higher levels of abstraction of ontology changes such as composite and complex ontology changes are not considered at all in that approach.

4.1.3 Semantics of Change

The semantics of change phase is the phase in the ontology evolution process that enables the resolution of ontology changes in a systemantic manner by ensuring the consistency of the entology. In the following we provide an overview of various notions of consistency and approaches for the realization of the changes.

Consistency

The goal of the semantics of change phase is to ensure that the application of ontology changes results in an ontology conforming to its consistency model. [Sto04] defines consistency as: "A single ontology OI is defined to be consistent with the respect to its model if and only if it preserves the constraints defined for underlying ontology model." For example, in the KAON ontology model, the consistency of an ontology is defined using a set of constraints, called invariants. These invariants state for example that the concept hierarchy has to be a directed acyclic graph.

The OWL ontology language also defines structural constraints on valid ontologies for the various fragments. In particular, [BvHH$^+$] defines the following constraints for OWL DL:

- OWL DL requires a pairwise separation between classes, datatypes, datatype properties, object properties, annotation properties, ontology properties (i.e., the import and versioning stuff), individuals, data values and the built-in vocabulary. This means that, for example, a class cannot be at the same time an individual.
- In OWL DL the set of object properties and datatype properties are disjoint. This implies that the following four property characteristics: inverse of, inverse functional, symmetric, and transitive can never be specified for datatype properties
- OWL DL requires that no cardinality constraints (local nor global) can be placed on transitive properties or their inverses or any of their superproperties.
- Annotations are allowed only under certain conditions.
- Most RDF(S) vocabulary cannot be used within OWL DL. See the OWL Semantics and Abstract Syntax document [PSHH] for details.
- All axioms must be well-formed, with no missing or extra components, and must form a tree-like structure.
- Axioms (facts) about individual equality and difference must be about named individuals.

However, regarding the consistency of Description Logics, we can also provide a model-theoretic definition. [PSHH] defines consistency as follows:

> A collection of abstract OWL ontologies and axioms and facts is consistent with respect to datatype map D iff there is some interpretation I with respect to D such that I satisfies each ontology and axiom and fact in the collection.

Please note, that with this model-theoretic definition, a DL-ontology can only become inconsistent by adding axioms: If a set of axioms is satisfyable, it will still be satisfyable when any axiom is deleted. Other approaches, e.g. adding constraints, would require non-monotonic reasoning, which are beyond the scope of OWL ontologies.

We now provide a typical example of introducing model-theoretic inconsistencies in the evolution of a DL ontology. Suppose, we start out with a very simple ontology about animals:

$bird \sqsubseteq animal$ (All Birds are animals)

$bird \sqsubseteq fly$ (All birds can fly) As the ontology evolves and more facts are added:

$dove \sqsubseteq bird$ (All doves are birds)

But as we add the following facts about penguins.

$penguin \sqsubseteq bird$ (All penguins are birds)

$penguin \sqsubseteq \neg fly$ (Penguins are not flying animals)

the ontology becomes inconsistent, as the concept penguin is unsatisfiable. Now, there may be many ways how to resolve this inconsistency. One possibility would be to reject either the change $penguin \sqsubseteq bird$or $penguin \sqsubseteq \neg fly$ from the set of changes. However, both of these axioms are correct, and the user may not be happy with this decision. The most intuitive one may be to retract the axiom $bird \sqsubseteq fly$, but also this may not satisfy the user.

[Sto04] describes and compares two approaches to verify ontology consistency:

1. a posteriori verification, where first the changes are executed, and then the updated ontology is checked whether it satisfies the consistency constraints
2. a priori verification, which defines a respective set of preconditions for each change. It must be proven that, for each change, the consistency will be maintained if (1) an ontology is consistent prior to an update and (2) the preconditions are satisfied.

Realization

[SMMS02] and [SMSS03] describe two approaches for the realization of the semantics of change, a procedural and a declarative one, respectively. In both these approaches, the KAON ontology model is assumed. The two approaches were adopted from the database community and followed to ensure the consistency in pursuing this semantics of change problem [FGM00]:

1. Procedural approach - this approach is based on the constraints, which define the consistency of a schema, and definite rules, which must be followed to maintain constraints satisfied after each change;
2. Declarative approach - this approach is based on the sound and complete set of axioms (provided with an inference mechanism) that formalises the dynamics of the evolution.

In [SMMS02] (procedural approach) the authors focus on providing the user with capabilities to control and customize the realization of the semantics of change. They introduce the concept of an evolution strategy encapsulating policy for evolution with respect to user s requirements. To resolve a change, the evolution process needs to determine answers at many *resolution points* – branch points during change resolution where taking a different path will produce different results. Each possible answer at each resolution point is an *elementary evolution strategy*. A common policy consisting of a set of elementary evolution strategies, each giving an answer for one resolution point, is an *evolution strategy* and is used to customize the ontology evolution process. Thus, an evolution strategy unambiguously defines the way how elementary changes will be resolved.

Typically a particular evolution strategy is chosen by the user at the start of the ontology evolution process.

In [SMSS03] (declarative approach) the authors present an approach to model ontology evolution as reconfiguration-design problem solving. The problem is reduced to a graph search where the nodes are evolving ontologies and the edges represent the changes that transform the source node into the target node. The search is guided by the constraints provided partially by user and partially by a set of rules defining ontology consistency. In this way they allow a user to specify an arbitrary request declaratively and ensure its resolving.

4.1.4 Change Implementation

We analyze the different roles of the change implementation phase: (i) to inform an ontology engineer about all consequences of a change request, (ii) to apply all the (required and derived) changes and (iii) to keep track about performed changes.

Change Notification

In order to avoid performing undesired changes, a list of all implications to the ontology and dependent artefacts should be generated and presented to the ontology engineer, who should then be able to accept or abort these changes.

Change Application

The application of a change should have transactional properties, i.e. (A) Atomicity, (C) Consistency, (I) Isolation, and (D) Durability. The approach of [Sto04] realizes this requirement by the strict separation between the request specification and the change implementation. This allows to easily treat the set of change operations as one atomic transaction, as all the changes are applied at once.

Change Logging

There are various ways to keep track of the performed changes. [Sto04] proposes an *evolution log* based on an *evolution ontology* for the KAON ontology model. The evolution ontology covers the various types of changes, dependencies between changes (causal dependencies as well as ordering), as well as the decision making process.

4.1.5 Change Propagation

[MMS03] presents an approach for evolution in the context of dependent and distributed ontologies. The authors define the notion of *Dependent Ontology Consistency*: A dependent ontology is consistent if the ontology itself and all its included ontologies, observed alone and independently of the ontologies in which are reused, are single ontology consistent. *Push-based* and *Pull-based* approaches for the synchronization of dependent ontologies are compared. The authors follow a push-based approach for dependent ontologies on one node and present an algorithm for dependent ontology evolution.

Further, for the case of multiple ontologies on multiple nodes, the authors define *Replication Ontology Consistency* (An ontology is replication consistent if it is equivalent to its original and all its included ontologies (directly and indirectly) are replication consistent.) and Here, for the synchronization between originals and replicas, the authors follow a pull-based approach.

Evolution in modular and distributed ontologies

One particular challenge of change propagation arises in the context of modularization. [SK03] concentrates on the benefits of modular ontologies with respect to local containment of terminological reasoning. The authors define an architecture for modular ontologies that supports local reasoning by compiling implied subsumption relations. They further address the problem of guaranteeing the integrity of a modular ontology in the presence of local changes. They propose a strategy for analyzing changes and guiding the process of updating compiled information.

The authors address the problem of guaranteeing the integrity of a modular ontology in the presence of a local change. They propose a strategy for analysing changes and guiding the process of updating compiled information. Ontology modules are connected by conjunctive queries. In order to make local reasoning independent of other modules, the authors use a knowledge compilation approach. The result of each mapping query is computed off-line and added as an axiom to the ontology module using that result. Once a query has been compiled, the correctness of reasoning can only be guaranteed as long as the concept hierarchy of the queried ontology module does not change. The authors propose a heuristic change detection mechanism that analyses changes with respect to their impact on the concept hierarchy. The set of changes they consider is not complete, as they focus only on changes regarding the concept hierarchy.

4.2 Ontology Versioning

In [HH00] the authors discuss the problems associated with managing ontologies in distributed environments such as the Web. They present SHOE, a web-based knowledge

representation language that supports multiple versions of ontologies. SHOE is described in the terms of a logic that separates data from ontologies and allows ontologies to provide different perspectives on the data. The paper presents the features of SHOE that address ontology versioning, the effects of ontology revision on SHOE web pages, and methods for implementing ontology integration using SHOE s extension and version mechanisms.

In [KF01] the authors discuss the problem of ontology versioning based on work done in database schema versioning and program interface versioning. They also propose building blocks for the two important aspects of a versioning mechanism: ontology identification and change specification.

[KKOF02] discusses OntoView, a web-based change management system for ontologies. OntoView provides a transparent interface to different versions of ontologies, by maintaining not only the transformations between them, but also the conceptual relation between concepts in different versions. It uses several rules to find changes in ontologies and it visualizes them — and some of their possible consequences — in the file representations. The user is able to specify the conceptual implication of the differences, which allows the interoperability of data that is described by the ontologies. The paper describes the system and presents the mechanism that we used to find and classify changes in RDFS / DAML ontologies. It also shows how users can specify the conceptual implication of changes to help interoperability.

4.3 Evolution and Versioning in Database Systems

According to [SR03], schema evolution has three well-defined and inter-related activities:

1. Core schema evolution, which includes identifying and incorporating changes to the schema while preserving the consistent state of the schema as well as propagating the changes to the data associated with the schema,

2. Version management, which deals with the management of different versions of a schema introduced by schema changes

3. Application management, which examines how applications dependent on the schema may continue to work.

The consistent state of a schema is typically defined using a formal structure such as a graph.

Whereas schema evolution in relational databases is only poorly supported, with the appearance of object-oriented database systems, schema evolution became a research issue. Orion very early prototype of an object-oriented system that provides support for schema evolution is ORION[BKKK87]. An important part of this work is the development of a formal taxonomy of schema changes and a framework for managing schema

changes in object-oriented systems. The semantics of each schema change is examined and a set of invariant properties of the schema is proposed which must be preserved across schema changes.

Object-oriented schema evolution is more relevant for the ontology evolution due to two reasons:

- the object-oriented database models provide a semantically richer model than the relational database system and, therefore, can be considered as the extension of the relation database evolution;
- the object-oriented database models are more similar to the ontology models due to the complex inheritance hierarchies.

In [NK03] the authors compare evolution of evolution of ontologies with evolution of database schemas. They state that the similarities between database-schema evolution and ontology evolution allow to build on the extensive research in schema evolution. They also identify important differences between database schemas and ontologies. The differences stem from different usage paradigms, the presence of explicit semantics, and different knowledge models. A lot of problems that existed only in theory in database research come to the forefront as practical problems in ontology evolution. These differences have important implications for the development of ontology evolution frameworks: The traditional distinction between versioning and evolution is not applicable to ontologies. There are several dimensions along which compatibility between versions must be considered. The set of change operations for ontologies is different.

[AFM03] presents two perspectives on modelling dynamic information using description logics: In a first part, the authors present a general temporally enhanced conceptual data model able to represent time varying data. In a second part, they introduce an object-oriented conceptual data model enriched with schema change operators, which are able to represent the explicit temporal evolution of the schema while maintaining a consistent view on the instantiated (static) data. Both conceptual data models and their inference problems are encoded in Description Logics.

The problem of the schema evolution and of the schema versioning support has been extensively studied in relational and database papers. [Rod95] provides an excellent survey of the main issues concerned. A semantic approach to the specification and management of object-oriented databases with evolving schemata in introduced in [FGM00]. The authors formalize a generic object-oriented model for the schema versioning and evolution, define the semantics of schema changes and show how interesting reasoning tasks can be supported. This approach is very similar to the declarative approach for the semantics of change in [Sto04] since both of them can deal with arbitrary complex changes and allow the formal checking of the evolution. A sound and complete axiomatic model for dynamic schema evolution in object-based systems is described in [Pz97]. This is the first effort in developing a formal basis for the schema evolution research. The approach takes

into account the key features of types and inheritance. The model can infer all schema relationships from two sets associated with each type.

4.3.1 Mapping Evolution

Mappings are an established paradigm to achieve interoperability in information systems applications. Mappings to translate data from one representation to another. As in dynamic environments data sources may change not only their data but also their schemas, their semantics, and their query capabilities. Such changes must be reflected in the mappings. Mappings left inconsistent by a schema change have to be detected and updated. Possible application scenarios for evolving mappings include, but are not limited to, data integration, model management, or local mappings between peers in Peer-to-Peer information management systems. Related to schema versioning, it is also possible to treat the new version of a schema as a "view" by providing the corresponding mapping.

[VMP03] presents a framework and a tool (ToMAS) for automatically adapting mappings as schemas evolve. Our approach considers not only local changes to a schema, but also changes that may affect and transform many components of a schema. The authors consider a comprehensive class of mappings for relational and XML schemas with choice types and (nested) constraints. Their algorithm detects mappings affected by a structural or constraint change and generates all the rewritings that are consistent with the semantics of the mapped schemas. The approach explicitly models mapping choices made by a user and maintains these choices, whenever possible, as the schemas and mappings evolve. The algorithms have been implemented in a mapping management and adaptation tool ToMAS.

[MP02] presents an approach to schema evolution, which combines the activities of schema integration and schema evolution into one framework. It builds on previous work on a general framework to support schema transformation and integration in heterogeneous database architectures, which relies on the hypergraph-based data model (HDM) as a common data model. In this paper the authors show how this framework also supports evolution of source schemas, allowing the global schema and the query translation pathways to be easily repaired, as opposed to having to be regenerated, after changes to source schemas.

4.4 Evolution and Versioning for Other Paradigms

4.4.1 Maintenance of Knowledge-Base Systems

Research in the ontology evolution can also benefit from the many years of research in knowledge-based system evolution [Men99]. There are a vast number of techniques (e.g. knowledge refinement, theory revision, validation and verification, etc.) for assisting the

development of knowledge-based systems. They follow the paradigm of detecting problems in a knowledge-based system and suggesting repairs. Most of them attempt to correct errors when a knowledge-based system is being developed.

A common technique for the knowledge maintenance is to reflect over dependencies between knowledge elements. The question is how to represent the dependencies between them. There are three different approaches: (i) procedural approach, (ii) logical approach, and (iii) network approach. All of them are based on the dependency graph analysis. The differences between these approaches are discussed in [MD00].

Chapter 5

Conclusion and Recommendations

We have presented an overview of state of the art in ontology evolution. We have shown a variety of approaches that concern the evolution process, frameworks, methods and tools. Many of these approaches build on top of or extend the work that has been done in the database and related communities.

However, there are still many open research questions, among them:

Language-independent ontology evolution: The existing ontology evolution approaches heavily depend on the underlying ontology language. One way to address this problem is to model all aspects of the ontology evolution declaratively. This abstraction may assist in the design of a language independent ontology evolution system.

Request specification: A declarative language for the specification of a request for a change would be desirable. It would allow expressing the ontology changes and constraints in a single framework, and, thus, would allow to reason about interactions between the two. This language would differ from the existing ontology query languages, which are only used for the retrieval of the data from an ontology. It would extend these languages by incorporating the modifications, as well.

Ontology dependency: Future research can be directed towards providing more ways for working with multiple ontologies. There are various dependency forms, such as ontology mapping, ontology merging, ontology alignment and ontology integration. For example, the ontology mapping relates similar (according to some metric) concepts and relations from different sources to each other; the ontology merging creates a new ontology from two or more existing ontologies with overlapping parts. Each of these dependency forms puts different requirements on the evolution between dependent ontologies. Some of them can be resolved by introducing a special meta-ontology that captures relationships between entities from different ontologies. For example, to set up the mapping

between ontologies, the mapping ontology might be defined. This ontology should contain the equal property that can be used for establishing equivalence between concepts from different ontologies. To support the evolution of the instantiation of this ontology, the ontology evolution approach has to be extended in two ways. Firstly, the set of consistency constraints has to be extended by taking into account the semantics of the mapping ontology. Secondly, the set of changes has to be extended with the more complex changes that can be applied to these mappings. Finally, the evolution support has to take into account that concepts and properties from the ontologies between which the mapping is established are considered as instances in the ontology that describes these mappings.

In the context of this project, the focus of the research will be on developing a framework and methods for the evolution of OWL-based ontologies and their application for data integration scenarios in the presence of heterogenous evolving data sources.

Bibliography

[AFM03] Alessandro Artale, Enrico Franconi, and Federica Mandreoli. Description logics for modeling dynamic information. In *Logics for Emerging Applications of Databases*, 2003.

[BHGS01] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: A reason-able ontology editor for the semantic web. In *KI-2001: Advances in Artificial Intelligence*, LNAI 2174, pages 396–408. Springer, 2001.

[BKKK87] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 311–322. ACM Press, 1987.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 2001(5), 2001. available at http://www.sciam.com/2001/0501issue/0501berners-lee.html.

[BvHH+] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language reference. `http://www.w3.org/TR/owl-ref/`.

[C+03] Per Cederqvist et al. *The CVS manual - Version Management with CVS*. Network Theory Limited, 2003.

[CFF+98] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. Rice. OKBC: A programmatic foundation for knowledge base interoperability. In *AAAI/IAAI*, pages 600–607, 1998.

[EGH+04a] M. Ehrig, T. Gabel, P. Haase, Y. Sure, C. Tempich, and J. Voelker. Data manual - initial version. SEKT informal deliverable 7.1.1.b, Institute AIFB, University of Karlsruhe, 2004.

[EGH+04b] M. Ehrig, T. Gabel, P. Haase, Y. Sure, C. Tempich, and J. Voelker. Use cases - initial version. SEKT informal deliverable 7.1.1.a, Institute AIFB, University of Karlsruhe, 2004.

[FGM00] Enrico Franconi, Fabio Grandi, and Federica Mandreoli. A semantic approach for schema evolution and versioning in object-oriented databases. In *Proceedings of the First International Conference on Computational Logic*, pages 1048–1062. Springer-Verlag, 2000.

[GPAFL+02] A. Gómez-Pérez, J. Angele, M. Fernandéz-López, V. Christophides, A. Stutt, Y. Sure, et al. A survey on ontology tools. OntoWeb deliverable 1.3, Universidad Politecnia de Madrid, 2002.

[GSV04] T. Gabel, Y. Sure, and J. Voelker. KAON – ontology management infrastructure. SEKT informal deliverable 3.1.1.a, Institute AIFB, University of Karlsruhe, 2004.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis. Mathematical Foundations*. Springer Verlag, 1999.

[HEHS04] P. Haase, M. Ehrig, A. Hotho, and B. Schnizler. Personalized information access in a bibliographic peer-to-peer system. In *Proceedings of the AAAI Workshop on Semantic Web Personalization, 2004*, JUL 2004.

[HH00] Jeff Heflin and James A. Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 443–449. AAAI Press / The MIT Press, 2000.

[HH02] I. Horrocks and J. A. Hendler, editors. *Proceedings of the First International Semantic Web Conference: The Semantic Web (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science (LNCS)*, Sardinia, Italy, 2002. Springer.

[KF01] Michel Klein and Dieter Fensel. Ontology versioning for the Semantic Web. In *Proceedings of the First International Semantic Web Working Symposium (SWWS)*, pages 75–91, Stanford University, California, USA, July 30 – August 1, 2001.

[KFKO02] Michel Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology versioning and change detection on the web. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, number 2473 in LNCS, page 197 ff, Sigüenza, Spain, October 1–4, 2002.

[KK04] FZI Karlsruhe and AIFB Karlsruhe. KAON the Karlsruhe ontology and semantic web framework — developer's guide for KAON 1.2.7, 2004.

[KKOF02] Michel Klein, Atanas Kiryakov, Damyan Ognyanov, and Dieter Fensel. Finding and characterizing changes in ontologies. In *Proceedings of the*

21st International Conference on Conceptual Modeling (ER2002), number 2503 in LNCS, pages 79–89, Tampere, Finland, October 7–11, 2002.

[Kle04] Michel Klein. *Change Management for Distributed Ontologies*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

[KN03] Michel Klein and Natalya F. Noy. A component-based framework for ontology evolution. In *Proceedings of the Workshop on Ontologies and Distributed Systemsm, IJCAI '03*, Acapulco, Mexico, August9, 2003. Also available as Technical Report IR-504, Vrije Universiteit Amsterdam.

[MD00] T.J. Menzies and J. Debenham. Expert systems maintenance. *Encyclopedia of Computer Science and Technology*, 47(27):35–54, 2000. Available from `http://tim.menzies.com/pdf/00cst.pdf`.

[Men99] Tim Menzies. Knowledge maintenance: The state of the art. *The Knowledge Engineering Review*, 14(1), 1999.

[MMS03] Alexander Maedche, Boris Motik, and Ljiljana Stojanovic. Managing multiple and distributed ontologies in the semantic web. *VLDB Journal*, 12(4):286–302, 2003.

[MP02] Peter McBrien and Alexandra Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, pages 484–499. Springer-Verlag, 2002.

[MV01] A. Maedche and R. Volz. The ontology extraction and maintenance framework text-to-onto. In *Proceedings of the ICDM'01 Workshop on Integrating Data Mining and Knowledge Management*, 2001.

[NFM00] N. Noy, R. Fergerson, and M. Musen. The knowledge model of Protégé-2000: Combining interoperability and flexibility. volume 1937 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 17–32, Juan-les-Pins, France, 2000. Springer.

[NK03] Natalya F. Noy and Michel Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 5, 2003. in press.

[OK02] Damyan Ognyanov and Atanas Kiryakov. Tracking changes in rdf(s) repositories. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 373–378. Springer-Verlag, 2002.

[PK97] Anne Pons and Rudolf R. Keller. Schema evolution in object databases by catalogs. In *Proceedings of the International Database Engineering and*

Applications Symposium (IDEAS'97), Montreal, Canada, pages 368–376, 1997.

[PSHH] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax. `http://www.w3.org/TR/2004/REC-owl-semantics-20040210/`.

[Pz97] Randel J. Peters and M. Tamer Özsu. An axiomatic model of dynamic schema evolution in objectbase systems. *ACM Trans. Database Syst.*, 22(1):75–114, 1997.

[Rod95] John F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393, 1995.

[SAS03] Y. Sure, J. Angele, and S. Staab. OntoEdit: Multifaceted inferencing for ontology engineering. *Journal on Data Semantics*, LNCS(2800):128–152, 2003.

[SEA+02] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. OntoEdit: Collaborative ontology development for the semantic web. In Horrocks and Hendler [HH02], pages 221–235.

[SHG03] N. Stojanovic, J. Hartmann, and J. Gonzalez. Ontomanager - a system for usage-based ontology management. In *In Proceedings of FGML Workshop. Special Interest Group of German Information Society (FGML - Fachgruppe Maschinelles Lernen der GI e.V.)*, 2003.

[SK03] Heiner Stuckenschmidt and Michel Klein. Integrity and change in modular ontologies. In *Proceedingso of the 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August 2003.

[SMMS02] Ljiljana Stojanovic, Alexander Mädche, Boris Motik, and Nenad Stojanovic. User-driven ontology evolution management. In *European Conf. Knowledge Eng. and Management (EKAW 2002)*, pages 285–300. Springer-Verlag, 2002.

[SMSS03] Ljiljana Stojanovic, Alexander Maedche, Nenad Stojanovic, and Rudi Studer. Ontology evolution as reconfiguration-design problem solving. In *KCAP 2003*, pages 162–171. ACM, OCT 2003.

[SR03] Ganesan Shankaranarayanan and Sudha Ram. Research issues in database schema evolution - the road not taken. Technical Report 2003-15, The University of Arizona, 2003.

[ST04] Y. Sure and C. Tempich. State of the art in ontology engineering methodologies. SEKT informal deliverable 7.1.2, Institute AIFB, University of Karlsruhe, 2004.

[Sto04] Ljlijana Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe, 2004.

[VMP03] Yannis Velegrakis, Renee J. Miller, and Lucian Popa. Mapping adaptation under evolving schemas. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, 2003.