GECAD

Ontology Mapping Evolution

2007 / 2008

Hélio Artur Mendes Martins

Ontology Mapping Evolution

Hélio Artur Mendes Martins

Advised by **Nuno Silva**

GECAD

MESTRADO EM ENGENHARIA INFORMÁTICA

Tecnologias do Conhecimento e Decisão

2007 / 2008

October 2008

«To my Parents»

Agradecimentos

O leque de pessoas e instituições que, directa ou indirectamente, desempenharam um papel fundamental para que este processo de pesquisa, análise e elaboração deste trabalho de mestrado, que me comprometi realizar, tivesse sido concluído com êxito é vasto, o que me enche de orgulho e satisfação, pelo que gostaria desde já lamentar qualquer eventual esquecimento.

Começo por agradecer **ao meu orientador**, Doutor Nuno Alexandre Pinto Silva por toda a ajuda, compreensão e disponibilidade que demonstrou sempre, pelos concelhos e críticas construtivas e incentivo para fazer sempre o meu melhor;

Ao GECAD e em especial ao Doutor Carlos Ramos, pela oportunidade que me concedeu a realizar o trabalho, e nos meios disponibilizados para o efeito;

Aos Projectos **EDGAR** (POSI/EIA/61307/2004) e **Coalesce** (PTDC/EIA/74417/2006) financiados pela MCTES-FCT no contexto dos quais foi realizado esta dissertação de mestrado, prestando por isso um apoio institucional importante para a sua conclusão;

Aos meus **amigos ISEPianos**: Hernâni, Silvino, Harry, Bruno, Nazu, Zex, Izy Love, Sandra, Faby, Hinze, Massa, por me terem aturado estes anos todos, mesmo nos meus períodos de mau humor;

Aos meus **colegas do GECAD**, em especial ao meu colega de grupo Paulo Santos, aos **funcionários do ISEP**, em especial ao Daniel e o segurança Dias por estarem sempre presente comigo nas minhas horas de trabalho;

Agradeço aos meus **ex companheiros de casa**: Heleno, Danilo, Nedil, Ima, Pifas, Pox, Tinovsky, Helter o grande companheirismo e palavras de incentivo;

Agradeço aos meus **grande amigos**: Valério, Heitor, Décio, Kinito, Odair, July, Vi, Rufo, Nhox pelos grandes momentos que passamos juntos.

As minhas **grandes amigas**: Liliana (“Nha orgulho”), Sheila (“Nha mana”), Rix, Helga, Taus, Evanilde, Rox, Suzy, Indira, Sónia e Daniela pela amizade, carinho e ternura que sempre me demonstraram;

Um especial obrigado ao meu grande colega e amigo **Danielson Alves** por esses anos todos de companheirismo, amizade, conselhos e atenção; a **Luiza Gabriel** por ser o meu braço esquerdo e a minha fonte de inspiração durante este trabalho; ao **Jorge Santos** pelos grandes momentos de cultura geral proporcionados;

Por último, mas não menos importante, queria agradecer à **minha família** por estes anos de apoio incondicional à minha vida académica e pessoal. Ao **meu Pai** Belmiro Pereira Martins, à **minha mãe** Amália Faustino Mendes, ao **meu irmão** Hélder Belmiro Mendes Martins e a **minha irmã** Edith do Rosário Mendes Martins, ao **meu primo** Amândio, ao **meu sobrinho** João de Sarafin, um muito obrigado.

Resumo Alargado

Introdução

Hoje em dia, o conceito de ontologia (Especificação explícita de uma conceptualização [Gruber, 1993]) é um conceito chave em sistemas baseados em conhecimento em geral e na Web Semântica em particular. Entretanto, os agentes de software nem sempre concordam com a mesma conceptualização, justificando assim a existência de diversas ontologias, mesmo que tratando o mesmo domínio de discurso. Para resolver/minimizar o problema de interoperabilidade entre estes agentes, o mapeamento de ontologias provou ser uma boa solução. O mapeamento de ontologias é o processo onde são especificadas relações semânticas entre entidades da ontologia origem e destino ao nível conceptual, e que por sua vez podem ser utilizados para transformar instâncias baseadas na ontologia origem em instâncias baseadas na ontologia destino.

Motivação

Num ambiente dinâmico como a Web Semântica, os agentes alteram não só os seus dados mas também a sua estrutura e semântica (ontologias). Este processo, denominado evolução de ontologias, pode ser definido como uma adaptação temporal da ontologia através de alterações que surgem no domínio ou nos objectivos da própria ontologia, e da gestão consistente dessas alterações [Stojanovic, 2004], podendo por vezes deixar o documento de mapeamento inconsistente. Em ambientes heterogéneos onde a interoperabilidade entre sistemas depende do documento de mapeamento, este deve reflectir as alterações efectuadas nas ontologias, existindo neste caso duas soluções: (i) gerar um novo documento de mapeamento (processo exigente em termos de tempo e recursos computacionais) ou (ii) adaptar o documento de mapeamento, corrigindo relações semânticas inválidas e criar novas relações se forem necessárias (processo

menos existente em termos de tempo e recursos computacionais, mas muito dependente da informação sobre as alterações efectuadas).

O principal objectivo deste trabalho é a análise, especificação e desenvolvimento do processo de evolução do documento de mapeamento de forma a reflectir as alterações efectuadas durante o processo de evolução de ontologias.

Contexto

Este trabalho foi desenvolvido no contexto do MAFRA Toolkit[1]. O MAFRA (MApping FRAmework) Toolkit é uma aplicação desenvolvida no GECAD[2] que permite a especificação declarativa de relações semânticas entre entidades de uma ontologia origem e outra de destino, utilizando os seguintes componentes principais:

- Concept Bridge – Representa uma relação semântica entre um conceito de origem e um de destino;
- Property Bridge – Representa uma relação semântica entre uma ou mais propriedades de origem e uma ou mais propriedades de destino;
- Service – São aplicados às Semantic Bridges (Property e Concept Bridges) definindo como as instâncias origem devem ser transformadas em instâncias de destino.

Estes conceitos estão especificados na ontologia SBO (Semantic Bridge Ontology) [Silva, 2004]. No contexto deste trabalho, um documento de mapeamento é uma instanciação do SBO, contendo relações semânticas entre entidades da ontologia de origem e da ontologia de destino.

Processo de evolução do mapeamento

O processo de evolução de mapeamento é o processo onde as entidades do documento de mapeamento são adaptadas, reflectindo eventuais alterações nas ontologias mapeadas, tentando o quanto possível preservar a semântica das relações semântica especificadas. Se as ontologias origem e/ou destino sofrerem alterações, algumas relações semânticas podem tornar-se inválidas, ou novas relações serão necessárias, sendo por isso este processo composto por dois sub-processos: (i) correcção de relações semânticas e (ii) processamento de novas entidades das ontologias.

O processamento de novas entidades das ontologias requer a descoberta e cálculo de semelhanças entre entidades e a especificação de relações de acordo com a ontologia/linguagem SBO. Estas fases ("similarity measure" e "semantic bridging") são implementadas no MAFRA Toolkit, sendo o processo (semi-) automático de mapeamento de ontologias descrito em [Silva, 2004].

[1] *http://mafra-toolkit.sourceforge.net/*

[2] *Grupo de Investigação em Engenharia do Conhecimento e Apoio a Decisão*

O processo de correcção de entidades SBO inválidas requer um bom conhecimento da ontologia/linguagem SBO, das suas entidades e relações, e de todas as suas restrições, i.e. da sua estrutura e semântica. Este procedimento consiste em (i) identificar as entidades SBO inválidas, (ii) a causa da sua invalidez e (iii) corrigi-las da melhor forma possível. Nesta fase foi utilizada informação vinda do processo de evolução das ontologias com o objectivo de melhorar a qualidade de todo o processo.

Conclusões

Para além do processo de evolução do mapeamento desenvolvido, um dos pontos mais importantes deste trabalho foi a aquisição de um conhecimento mais profundo sobre ontologias, processo de evolução de ontologias, mapeamento etc., expansão dos horizontes de conhecimento, adquirindo ainda mais a consciência da complexidade do problema em questão, o que permite antever e perspectivar novos desafios para o futuro.

Palavras-chaves

Ontologia; Evolução de Ontologia; Mapeamento de Ontologias; Evolução de Mapeamento de Ontologias.

Abstract

Systems or software agents do not always agree on the information being shared, justifying the use of distinct ontologies, even if corresponding to the same domain of application. To achieve interoperability, modern information systems often use declarative mapping specification as base for information exchanging between them. However, in dynamic environments like the Web and the Semantic Web, actors may change their data and also their structure and semantics (ontology).

In a heterogeneous environment, where interoperability among systems depends essentially upon mapping document, these ontology changes must be reflected accordingly in the semantic relations in the ontology mapping document.

The main goal of this project is to develop a solution for evolving/adapting the ontology mapping document when the mapped ontologies evolve. The process of ontology mapping evolution is defined as a process whereby entities of ontology mapping document is adapted regarding eventual changes in source and/or target ontologies, trying to preserve as much as possible the semantics of ontology mapping semantic relations.

This document presents the developed approach for ontology mapping evolution process.

Keywords: Ontology; Ontology Evolution; Ontology Mapping; Ontology Mapping Evolution.

Table of Contents

List of Figures

List of Tables

Notations

API	Application Program Interface
GECAD	Grupo de Investigação em Engenharia de Conhecimento e Apoio a Decisão
HTML	Hypertext Markup Language
KAON	Karlsruhe Ontology
OWL	Ontology Web Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

Chapter 1

Introduction

Currently, ontology is a key concept in knowledge-based systems in general and in the Semantic Web and E-Business in particular ([Berners-Lee, et al., 2001]). However, actors do not always agree on the information being shared, justifying the use of distinct ontologies, even if corresponding to the same domain of application. To solve/minimize the interoperability problem, ontology mapping proved to be an efficient solution. Ontology mapping is the process whereby semantic relations are defined between two ontologies at conceptual level, which in turn are applied at data level transforming source ontology instances into target ontology instances [Silva, 2004]. The result of the ontology mapping specification process is an ontology mapping document containing those semantic relations.

In dynamic environments like the Web and the Semantic Web, actors may change their data and also their structure and semantics (ontology). These ontology changes must be reflected accordingly in the semantic relations in ontology mapping document.

1.1 Motivation

In dynamic environments, ontologies may change (ontology evolution), causing the ontology mapping document to become invalid. In a heterogeneous environment, where interoperability among systems depends

essentially upon the ontology mappings, semantic relations must be adapted to reflect ontology evolution. Figure 1 illustrates the motivation scenario for this work:

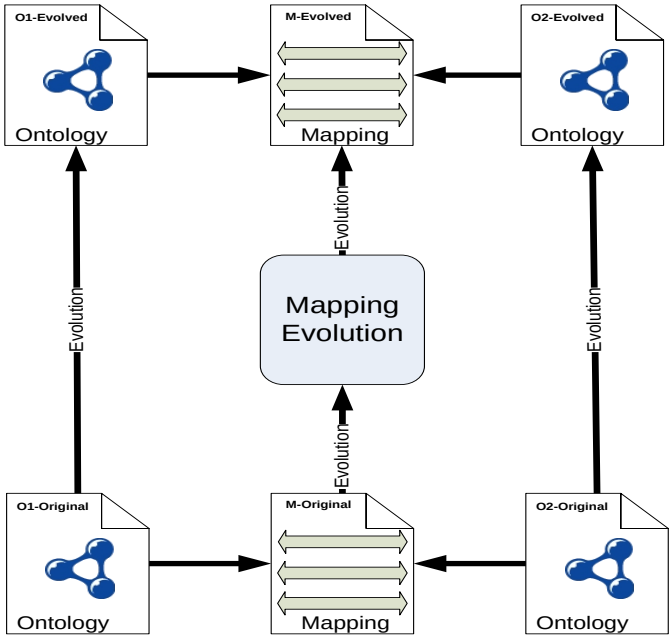

Figure 1 – Motivation Scenario

O1-Origianl and O2-Original – Represent source and target ontologies respectively;

M-Original between O1-Original and O2-Original – Represents the ontology mapping document containing semantic relations between entities of O1-Original ontology and O2-Original ontology;

O1-Evolved and O2-Evolved – Represent a new version of the source ontology (O1-Original) and a new version of the target ontology (O2-Original) respectively. Both or only one of the ontologies may evolve;

M-Evolved between O1-Original and O2-Original – Represents the ontology mapping document containing semantic relations between entities of O1-Evolved ontology and O2-Evolved ontology;

Mapping Evolution – It is the process that performs the evolution from Mapping-Original to Mapping-Evolved.

A potential solution for this problem is the generation of a completely new ontology mapping document, i.e. a new set of semantic relations between entities of new versions of source/target ontologies. However this process is very time and computational resources consuming.

Another solution is to adapt the original ontology mapping document, correcting invalid semantic relation and generating new (required) semantic relations according to ontology changes. This process is less time and computational consuming, but depends on the available information (and quality) about ontology changes. It is necessary to identify and classify the changes occurring in the mapped ontologies.

This project aims to develop a solution and system for supporting the evolution and adaptation of the ontology mapping document when the mapped ontologies evolve.

1.2 Context

This project was developed in the context of MAFRA Toolkit. MAFRA (MApping FRAmework) toolkit is an ontology mapping application developed in GECAD. This tool is based on MApping FRAmework, and allows the specification of semantic relations between source ontology entities and target ontology entities. For that, the following main building blocks are used:

- Concept Bridge – Represents a semantic relation between two ontology concepts (one source concept to one target concept);
- Property Bridge – Represents a semantic relation between two or more properties (at least one from source and at least one from the target ontology);
- Service – There are many services that can be applied to Property Bridges (e.g. copyAttribute, split, concat) which define how source instances will be transformed into target instances.

These concepts are described in the Semantic Bridge Ontology (SBO) proposed in [Silva 2004]. In MAFRA Toolkit context, the mapping document is an instantiation of the SBO. It contains a set of SBO entities (Concept Bridges, Property Bridges, etc.) relating source ontology entities to target ontology entities.

1.3 Organization

This work was developed in GECAD (Knowledge Engineering and Decision Support Research Center). GECAD ("Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão" - Knowledge Engineering and Decision Support Research Center) is an R&D centre settled at the School of Engineering – Polytechnic of Porto (ISEP/IPP), operating in the areas of Artificial Intelligence, Knowledge-Based Systems and Decision Support Systems.

The GECAD's mission is the promotion and development of scientific research in the Knowledge and Decision Sciences domains, having Information Technologies as support [GECAD 2008].

This work was developed in context of the Knowledge & Learning Technologies research line.

1.4 Thesis Overview

This work is organized as follow:

Chapter 1 – Introduction – This chapter introduces this work, presenting the motivations and the context where it was developed.

Chapter 2 – Ontology and Ontology Mapping. This chapter describes the background concepts of this work. Ontologies, Semantic Web, Ontology representation languages and Ontology Mapping concepts are presented. An overview about MAFRA Toolkit is also given.

Chapter 3 – Understanding Ontology Evolution. This chapter describes the state-of-the-art in ontology evolution process. The goal of this chapter is the description on how ontology evolution process works and how is managed. The knowledge acquired in this chapter is used during the rest of the work.

Chapter 4 – Ontology Mapping Evolution Process. This chapter presents the developed process of ontology mapping process.

Chapter 5 – Correct Invalid SBO Entity Using Ontology Evolution Information. This chapter presents the process of correcting invalid SBO entities using ontology evolution information.

Chapter 6 – Correct Invalid SBO Entity without Ontology Evolution Information. This chapter presents the process of correcting invalid SBO entities without ontology evolution information.

Chapter 7 – Conclusion. This chapter presents the conclusions, problems, limitations and future work.

Chapter 2

Ontology and Ontology Mapping

This chapter presents the ontology concept according to the most common definitions and the concept of the ontology mapping according to the SBO (Semantic Bridge Ontology) representation language.

2.1 Ontology

The most common definition for ontology is: "**Ontology is an explicit specification of a conceptualization**" [Gruber, 1993].

Despite it is only a few words long, this definition is the most used in artificial intelligence area, and the closest of consensus. Let us try understanding the meaning of the words in this definition: explicit, specification and conceptualization, proposed by [Studer, et al., 1998] .

- **Conceptualization** – Means an abstract and rational of domain model where are included identification of concepts and their descriptions, properties and relations between them;
- **Specification** – Means description of some domain of discourse, detailed, accurate, consistent, solid and meaningful;
- **Explicit** – Contrary to implicit, means the exteriorization of conceptualization. In other words, the representation of conceptualization in way that it can be used and understood by agents.

Another well know definition is "**Ontology is a formal specification of shared conceptualization**" presented by [Brost, 1997] four years late. This definition adds the "formal" and "shared" keywords:

- **Formal** – Means that ontology is readable, understandable and processable not only by people, but also by machines;
- **Shared** – Means that the ontology is accepted as consensus in a group.

Actually, ontology concept is used in several contexts like e-commerce, multi-agent system, natural language processing, Semantic Web, etc ([Bontcheva, et al., 2004], [Fensel, 2001], [Schulten, et al., 2003]).

2.2 Semantic Web

I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize [Berners-Lee, 1999].

This citation made by [Berners-Lee, 1999], represents the "*dream*" of the father of Web and HTML[3]. Semantic Web can be understood as an evolution of the conventional World Wide Web, where data can be easily accessed, understood and processed by users and machines [Wikipedia/SemanticWeb, 2008]. Semantic Web intends to add semantics to data. In other words, provides a formal structure which gives meaning to Web pages where software agents can work together. Computers need to have access to structured data (data and metadata) and a set of rules that helps in automatic reasoning process [Wikipedia/SemanticWeb, 2008].

Web pages built using Semantic Web philosophy will be able to be read by humans or by machines and can be graphically represented, read by browsers or made available to others technologies.

Figure 2 represents the layers of Semantic Web [Wikipedia/SemanticWeb, 2008] illustrated in the next few lines:

- **Unicode/URI** – Responsible for ensuring that used character set have the same meaning on remaining layers;
- **XML + NS + xmlschema** – This layer is responsible for data representation allowing interoperability between systems at syntactical (schema) level;
- **RDF + RDFSchema** – RDF allow representing knowledge in form of triples (subject-predicate-object). This is an easy and generic way for representing knowledge. RDF Schema (RDFS) allows to describe vocabulary about domain of discourse;

[3] *Hypertext Markup Language*

- **Ontology Vocabulary** – This layer increases the capacity to model semantics of domain of discourse by adding new complex relation between concepts;
- **Logic** – This layer enables the writing of rules;
- **Proof** – This layer executes the rules of logic layer;
- **Trust** – This layer provides mechanism for application whether to trust the given proof or not;
- **Digital Signature** – This orthogonal component is responsible for detecting alterations to documents.

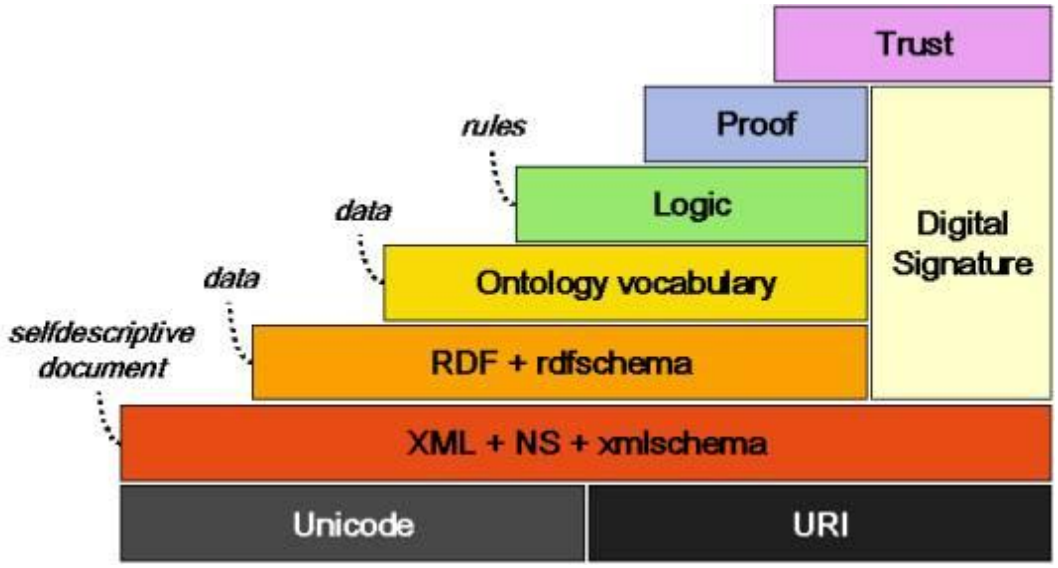

Figure 2 – Semantic Web layer cake

2.2.1 Why the Semantic Web

Due the huge amount of stored data available on the Internet, finding exactly what one wants and mostly process it, is more and more a difficult and time consuming task. Since this amount of data is not formally represented, software agents cannot access, understand and process it in deterministic time.

Semantic Web increases the precision and recall of search engines and others tools for automatic document processing. The basic idea is to create an environment where available data is described in an expressive way, and artificial agents can execute boring, time consuming and/or sophisticate tasks for us [Wikipedia/SemanticWeb, 2008].

2.2.2 Data Structure and Representation

Data structure and representation are main dimensions of the Semantic Web. Enabling data access to artificial agents' knowledge bases, the data need to be represented in a common representation format and language. This common representation provides interoperability between agents.

Representing semantics of data is related not only with content of a resource, but also about their nature and relations. Herewith, it is vital that data is represented in an expressive way so machines can understand its real meaning.

In the Semantic Web, the semantics of data is often represented using ontologies.

2.3 Representation Languages

Knowledge representation languages are not exclusive of the Semantic Web, and are being around for several decades (e.g. FLogic, KIF). Yet, with the advent of the Semantic Web a new important technological context occurred, proposing not only an important application scenario but also a source of relevant technology.

2.3.1 RDF – Resource Description Framework

RDF [W3C/RDF, 2004] is a language for representing information about resources in the Internet. The basic idea of RDF model is to make statements about Web resources[4] in the form of subject-predicate-object expressions, called triples. A triple is a statement where:

- **Subject** – represents the resource to classify. For example, http://w3.org/;
- **Predicate** – represents a property or attribute of resource and express the relationship between subject to object. For example: has title (attribute or relation to object);
- **Object** – represents the value of the predicate in the context of the subject. For example the string "World Wide Web Consortium".

This statement can be represented in the form of a graph:

Figure 3 – RDF graph representation example[5]

RDF is an URI-based language. In other words, all resources are identified by an URI[6]. Uniform Resource Identifier (URI) is a compact string of characters used to identify resources like Web pages, services, documents, images, etc. on the Internet.

A RDF File is a set of statements in the form of triples classifying and relating resources. These statements are based on terminology defined in schemas, where the semantic of resources to use is expressed.

[4] *http://en.wikipedia.org/wiki/Resource_(Web)*

[5] *RDF graph generated by RDF Validator (http://www.w3.org/RDF/Validator/)*

[6] *URI - Uniform Resource Identifier (http://en.wikipedia.org/wiki/Uniform_Resource_Identifier)*

Accordingly, for defining the semantics of terminology one has to develop schemas or even better, ontologies, using ontology representation languages such as RDFS and OWL.

2.3.2 RDFS – Resource Description Framework Schema

RDF Schema (RDFS) [W3C/RDFS, 2004] is a general-purpose language for representing information on the internet, capable to describe the vocabulary used in RDF statements. RDFS contains concepts like classes and properties that can be used to represent resources on the internet. Concepts like classes and properties are similar to those used on the object-oriented programming languages, but for defining the relation between them is quite different. In typical object-oriented approach (e.g. programming languages such as Java or C++), classes are defined with a set of properties. In RDFS, properties are first-order citizens, where domain and range (co-domain) are defined. In other words, which class property belongs to (domain), and which values it can take (range) is defined in the context of properties and not in the classes. For example, a property author can be defined, and add a Book and/or Movie domain and add Person as range.

To define ontologies in RDFS, vocabulary that contains keywords like rdfs:Class, rdfs:Resource, rdfs:Literal, rdfs:Datatype, rdfs:XMLLiteral, rdf:Property is used. Such vocabulary is defined in the http://www.w3.org/2000/01/rdf-schema namespace.

2.3.3 OWL – Ontology Web Language

OWL Web Ontology Language [W3C/OWL, 2004] is an extension of RDFS and overcomes some of its limitations. OWL is an adaptation of DAML+OIL language and is categorized in three sublanguages:

- **OWL Lite** – It is the simplest OWL version, including hierarchy and some restrictions;
- **OWL DL** – This sublanguage increases the expressivity, including logic and retains computational completeness (all conclusion are guaranteed to be computable) and decidability (all computations will finish in finite time);
- **OWL Full** – This sublanguage guarantees maximum expressiveness, but there is no computational guarantee.

OWL introduces new concepts like:

- **Equality/Inequality** – equivalent class, equivalent property, same as, different from, all different;
- **Property Characteristics** – inverse of, transitive, symmetric and functional property;
- **Property restrictions** – allValuesFrom, someValuesFrom, hasValue;
- **Cardinality restrictions** – minCardinality, maxCardinality, cardinality;
- **Classes** – intersectionOf, unionOf;
- **Data types** – xsd datatypes;
- **Versioning** – versionInfo, priorVersion, backwardCompatibleWith, inCompatibeWith, DeprecatedClass, DeprecatedProperty.

2.3.4 Summary

In this work, all ontology representation language was taking into account. However, due the differences between them, ontologies in different languages cannot be mapped unless they are raised to the same representation level.

2.4 Ontology Mapping

Ontology mapping is the process whereby semantic relations are defined between two ontologies at the conceptual level, which in turn are applied at data level, transforming source ontology instances into target ontology instances [Silva, 2004].

In the Semantic Web, where information is distributed and heterogeneous, use of distinct ontologies even for the same domain, may turn the exchange of information between software agents more difficult. The IEEE dictionary [IEEE 90, 1990] defines interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged". For solving/minimizing the interoperability problem, ontology mapping proved to be an efficient solution ([Madhavan, et al., 2001], [Rahm, et al., 2001], [Euzenat, et al., 2007], [Silva, 2004]).

The output of the ontology mapping process is a formal mapping document containing semantic relations between source ontology entities to target ontology entities. Ontology mapping process is not a trivial process, requiring a deep understanding of both ontology conceptualizations and their semantic similarities. In fact:

- Concepts can have the same meaning but different names (e.g. Human and Person, Film and Movie);
- Concepts can have the same name, but different meaning (e.g. Jaguar in car domain, and Jaguar in animal domain, Keyboard in computer context and Keyboard in music context);
- Properties can have the same meaning, but with different names (e.g. sex and gender);
- Properties with the same name, but different meaning (e.g. properties with multiple domain: spouse with domain Man/Woman);
- Terms with the same meaning, but different writing (e.g. coulor and color);
- One source term has equivalence to more than one target term (e.g. spouse to wife, and spouse to husband);
- More than one source term correspond to one target term (e.g. first name + last name to name).

2.4.1 MAFRA Toolkit – MApping FRAmework toolkit

MAFRA (MApping FRAmework) toolkit is an ontology mapping tool that allows the specification of semantic relations between two (source to target) ontologies, and applies such relations in translating source ontology instances into target ontology instances. It implements a specific architecture of MAFRA (Figure 4).

2.4.1.1 Ontology Mapping Process Flow

Like many systems, the ontology mapping process, as the responsible for mapping document, adopts a cyclic approach, so it can fit the ontology mapping document manipulation requirements. Furthermore, the ontology mapping process exhibits the following characteristics:

- **Incremental** – The document is improved in every phase of the process;
- **Interactive** – The Human-being is the ultimate responsible for the flow of the process;
- **Continuous** – The Ontology mapping document can always be improved, especially if the automatic methods are applied.

Figure 4 represents the MAFRA architecture [Maedche, et al., 2002] emphasizing the horizontal and the vertical dimensions.

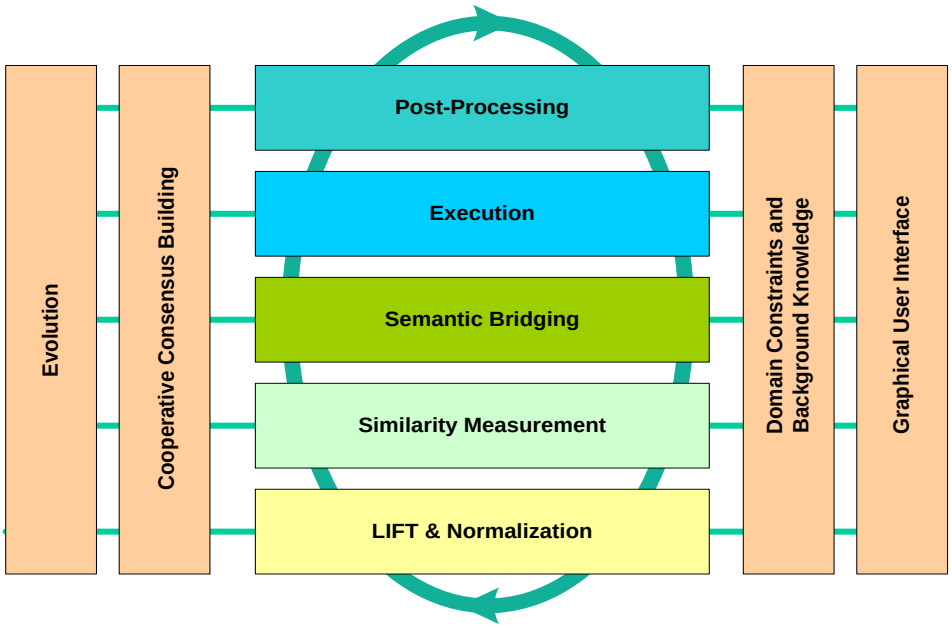

Figure 4 – MAFRA (MApping FRAmework) Architecture

2.4.1.2 Horizontal Dimension

- **LIFT & Normalization**: Raise all data to be mapped to the same representation level, and normalize the strings (tokenization, expansion of acronyms, etc.);
  - **Input** – Ontology and Instances;
  - **Output** – Normalized ontologies and instances;

- **Similarity Measurement**: Mapping discovery. Calculates the similarities according to several already proposed algorithms;
  - **Input** – Normalized ontologies and Instances;
  - **Output** – Matches;
- **Semantic Bridging**: Establishes expressions relating a set of source ontology entities with a set of target ontology entities through a transformation function;
  - **Input** – Normalized ontologies and Matches;
  - **Output** – Ontology Mapping Document;
- **Execution**: Transforms instances from source ontology to target ontology;
  - **Input** – Normalized source instances, normalized ontologies and Ontology Mapping Document;
  - **Output** – Normalized target instances (in a common representation model and language);
- **Post-Processing**: Takes the results of execution to check and improve the quality of transformation;
  - **Input** – Target instances;
  - **Output** – Validated and improved target instances.

2.4.1.3 Vertical Dimension

- **Evolution** – Aims to manage the ontology mapping document according to external factors like changes at source/target ontologies;
- **Cooperative Consensus Building** – Aims to support and promote consensus between two (or more) interoperability intervenients;
- **Domain Constraints and Background Knowledge** – Aims to improve the quality of similarity and automatic semantic bridging by using background knowledge like dictionaries of lexical terms and by using Domain Constraints like glossaries of domain acronyms and abbreviation;
- **Graphical User Interface** –Drives and constrains the user actions to promote better ontology mappings, while minimizing user efforts.

2.4.2 SBO – Semantic Bridge Ontology

SBO (Semantic Bridge Ontology) is the ontology used on MAFRA Toolkit to describe an ontology mapping document.

SBO describes the semantic relations domain of knowledge, providing not only a reasoning mechanism but also a representation and exchange mechanism of semantic relationships. SBO is a very simple and compact ontology, but its extensional structure based on transformation services allows its adoption in very distinct and complex scenarios. It specifies, classifies and describes the types of ontology mapping relations, inter relates them and provides other modeling constructs necessary to express ontology mapping documents.

SBO is fundamentally composed by two distinct but complementary concepts:

- **Service** – Represents the transformation capabilities existing in the ontology mapping system (e.g. Copy Relation, Copy Attribute, Currency Converter) that can be applied to the Semantic Bridges;
- **Semantic Bridge** – Represents the semantic relation between a set of source ontology entities and a set of target ontology entities. It encompasses all the information necessary to its correct and univocal interpretation at execution time (e.g. Concept Bridges, Property Bridges).

Figure 5 illustrates the relation between semantic bridges and services.

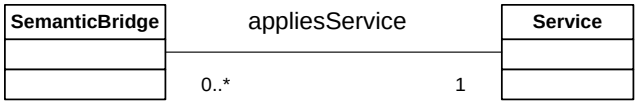

Figure 5 – Semantic Bridge and Service conceptual relation

Next sub-sections describe the main SBO concepts and relations.

2.4.2.1 Service

In MAFRA Toolkit context, a Service represents the transformation capabilities. A Service is applied to a Semantic Bridge and it defines how source instances are translated into target instances. Services are responsible for defining:

- **Cardinality** – Service defines how many arguments are allowed in source arguments and in target arguments (e.g. Concatenation Service allows many source arguments and just one target argument – n:1 cardinality);
- **Entity types** – Service defines the type of entity allowed in source and target arguments. The type of entities may be Concept, Relation, Attribute or Constant.

2.4.2.2 Semantic Bridges

Semantic Bridges represent semantic relations between source ontology entities and target ontology entities.

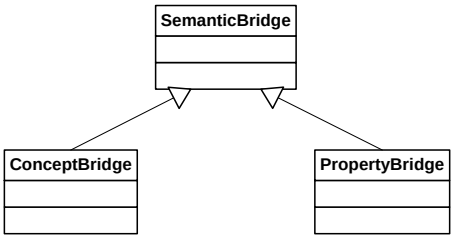

Figure 6 – SBO Taxonomy of Semantic Bridges

Following the same approach of ontology entities (Concepts and Properties), Semantic Bridge is specialized into Concept Bridge and Property Bridge (Figure 6).

2.4.2.3 Concept Bridge

Concept Bridge represents a semantic relation between source and target concepts. During the execution phase it means that the source concept instance will be transformed into a new target concept instance. Concept Bridge specializes the Semantic Bridge about two characteristics:

- **Cardinality is always 1:1** – Exactly one source concept is semantically related to exactly one target concept. Multiple Concept Bridges should be defined if one needs to relate two source concepts to the same target concept and vice-versa;
- **Always the same service applied** – The Copy Instance service is always applied to Concept Bridges, which during the execution phase is responsible for translating source concept instances into target concept instances.

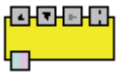

Figure 7 – Concept Bridge representation in MAFRA Toolkit GUI

Figure 7 represents Concept Bridge representation in MAFRA Toolkit GUI.

2.4.2.4 Property Bridge

Property Bridge represents a semantic relation between source and target properties. During the execution phase it means that the source property instance will be transformed into a new target property instance.

Contrary to Concept Bridges, the cardinality of Property Bridges can vary enormously according to the applied Service. The applied Service is responsible for:

- **Type of entities related** – The entities (source and target) related must be defined according to the service (e.g. the Concatenation Service must transform string attributes from source ontology into string attributes in the target ontology);
- **Cardinality of Property Bridge** – The cardinality of Property Bridge is defined by the applied Service (e.g. the cardinality of the concatenation Service is n: 1).

Figure 8 – Property Bridge representation in MAFRA Toolkit GUI

Figure 8 represents Property Bridge representation in MAFRA Toolkit GUI.

2.4.2.5 Alternative Bridge for Concept Bridge

This entity is not subclass of Semantic Bridge, in the sense that it is not responsible for the transformation of ontology instances. Instead, it is useful in the control of the execution flow. Alternative Bridge for Concept Bridge is responsible for the specification of disjoint relations between Concept Bridges. This relation between Concept Bridges arises from the need to conditionaly transform the instances of source concept into a set of instances of target concepts. In such cases, Concept Bridges are defined for each pair of source concept and the target concept. These Concept Bridges are grouped together into an Alternative Bridge that prevents the execution of more than one (disjoint) Concept Bridges.

2.4.2.6 Alternative Bridge for Property Bridge

Again, this entity is not sub class of Semantic Bridge. Alterative Bridge for Property Bridge is responsible for the specification of disjoint relations between Property Bridges. Similarly to the disjoint relation between Concept Bridges, a set of Property Bridges is often mutually disjoint. Only one Property Bridge defined in the Alternative Bridge is executed. Figure 9 represents Alternative Bridge representation in MAFRA Toolkit GUI.

Figure 9 – Alternative Bridge representation in MAFRA Toolkit GUI

2.4.2.7 HasBridge

The *hasBridge* entity is a relation between a Concept Bridge and a Property Bridge.

Concept Bridges are responsible for the creation of instances of target ontology concepts, which in turn will serve as containers or placeholders for instances of target ontology properties resulting from the execution of Property Bridges. This means that concept instances are composed by properties instances. During semantic bridging phase, this relation means that the properties of the source concept correspond to the target concept properties according to the Property Bridge. At execution time, the related Property Bridge will be executed for all properties instances defined in the source concept instances, giving rise to the target concept properties instances. Figure 10 represents the hasBridge relation in MAFRA Toolkit GUI.

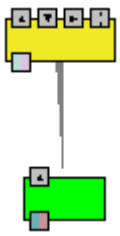

Figure 10 – The hasBridge relation representation in MAFRA Toolkit GUI

2.4.2.8 SubBridgeOf

The *subBridgeOf* relation is a hierarchical relation between Concept Bridges. We can define one Concept Bridge as sub bridge of (sub class of) another Concept Bridge, which means that all Property Bridges in the super Concept Bridge are inherited by the sub Concept Bridge. Figure 11 represents the subBridgeOf relation in MAFRA Toolkit GUI.

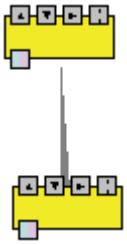

Figure 11 – The subBridgeOf relation representation in MAFRA Toolkit GUI

The sub bridge of relation is possible under specific circumstances, which are related to the sub class relation between the concepts of both Concept Bridges. A detailed explanation is available [Silva, 2004].

2.4.3 Evolution in MAFRA

The Evolution module of the MAFRA architecture is of special interest for this work. Repositories and their description tend to evolve in order to meet new requirements, to solve problems, and to increase performance [Silva, 2004]. As information sources evolve, interoperability features also evolve, adapting the semantic relations according to the information sources evolution. This module (Evolution) aims to manage the ontology mapping document according to external factors like changes in source and/or target ontologies.

This is not a trivial process, due to the variety of sources and consequences of changes and thus “cannot” be performed manually by an ontology engineer. Next chapter presents the state-of-the-art of this process.

Chapter 3

Understanding Ontology Evolution

Since the main goal of this work is to develop a solution for evolving/adapting ontology mapping document when related ontology evolves, it is really important to understand ontology evolution process and to be aware of the most approaches.

Ontology evolution can be defined as the timely adaptation of ontology to the arisen changes and the consistent management of these changes [Stojanovic, 2004]. Since a change in the ontology can cause inconsistencies in other parts of the ontology, as well as in the dependent artifacts, the ontology evolution has to be considered as a process. This is not a trivial process, due to the variety of sources and consequences of changes, and thus "cannot" be performed manually by an ontology engineer.

3.1 Ontology Evolution Dimensions

There are many approaches for ontology evolution process and there are many tools to support this process. According to the literature, the ontology evolution process includes three relevant dimensions for this work:

- **Change Representation** – One characteristic of ontology evolution approaches is how to represent ontology changes. First the changes that can be performed in ontologies need to be forecasted. After capturing these changes, they have to be represented in a suitable format in a formal way. The change representation depends on the used ontology language [Klein, 2004]. In order to improve the ontology evolution process, different level of abstraction is often used. This dimension allows understanding ontology changes and the relation between them;
- **Consistency Maintenance –** According to [IEEE 90, 1990], consistency is the degree of uniformity, standardization, and freedom from contradiction among the parts of a system or component. In [Stojanovic, 2004] the author states that ontology consistency can be considered as an agreement among ontology entities with the respect to the semantics of the underlying ontology language. Beyond agreement with the respect to the semantics of the ontology language, ontology consistency is related with ontology's purpose. This dimension allows understanding how ontology changes are performed in order to keep it consistent;
- **Change Storage** – The way change is stored is an important issue for ontology evolution systems. Changes need to be stored in a format that can be easily accessed and used in the future. Other (meta) information, like the relation between changes, owner of change request, etc. may be stored too. This dimension of the problem allows understanding how changes can be accessed, interpreted and understood by others systems.

In the following sections ontology evolution approaches are described. Besides an overview, each approach will be analyzed and characterized according to the previous three dimensions.

3.2 KAON Ontology Evolution Approach

This section describes the basics ideas of the KAON Evolution approach. This approach was proposed by [Stojanovic, 2004].

3.2.1 KAON ontology evolution overview

This approach proposed by [Stojanovic, 2004] is divided in six phases depicted in Figure 12.

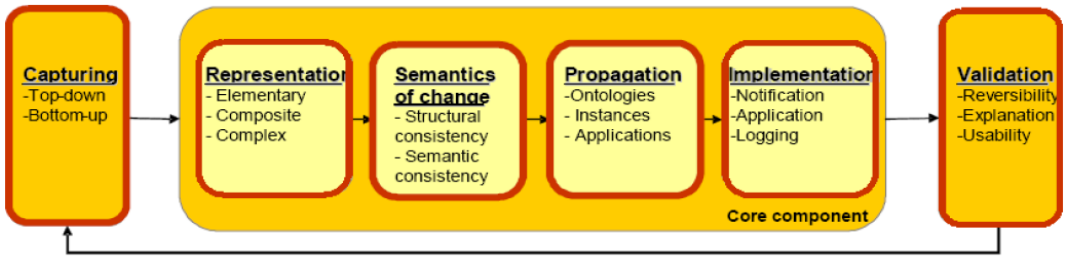

Figure 12 – The KAON Ontology evolution process approach

1. **Capturing** – The process begins with capturing changes either from explicit requirements or from the result of change discovery methods. The main goal of this phase is how to discover a change;
2. **Representation** – Changes are formally and explicitly represented as one or more ontology changes;
3. **Semantics of changes** – This phase prevents inconsistencies by computing additional changes that guarantee the transition of the ontology from one consistent state into another consistent state;
4. **Propagation** – The task of the change propagation phase is to automatically bring all dependent artifacts into a consistent state after an ontology update has been performed;
5. **Implementation** – During the change implementation phase:
    a. the ontology engineer is informed about all consequences of the requested change;
    b. the required and derived changes are applied in a transaction manner;
    c. the track about changes are saved.
6. **Validation** – In the change validation phase, the user evaluates the results and restarts the cycle if necessary.

3.2.2 Change Representation

Changes have to be represented in a suitable format. For representing those changes, KAON ontology evolution has three levels of abstractions (Figure 12):

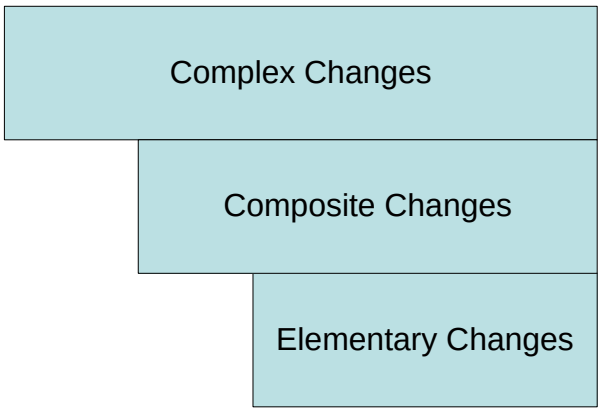

Figure 13 – The KAON Ontology evolution layers of abstraction

- **Elementary change** – Modifies (adds or removes) only one entity of the ontology model. It can be seen as an isolated modification of an ontology;
- **Composite change** – Modifies (creates, removes or changes) the neighborhood of an ontology entity. Composite changes specify coarse-grained changes. They are more powerful since an ontology engineer does not need to go through every step of the sequence of basic changes to achieve the desired effect;

- **Complex change** – Can be decomposed into any combination of at least two elementary and composite ontology changes. Complex changes encompass all "real-life" changes not included in the elementary and composite changes. Complex changes raise the level of abstraction and reusability even further.

3.2.2.1 Elementary Changes

Elementary changes modifies (adds or remove) only one entity of the ontology model, representing the ontology modifications at the lowest level of complexity. Therefore, they build the backbone of the ontology evolution system. Table 1 shows that ontology changes can be seen as *additive* and *subtractive* changes as is illustrated through the second and the third column, respectively.

Meta Entities/Meta Changes	Add	Remove
Concept	AddConcept	RemoveConcept
Concept Hierarchy	AddSubConcept	RemoveSubConcept
Property	AddProperty	RemoveProperty
Property Hierarchy	AddSubProperty	RemoveSubProperty
Property Domain	AddPropertyDomain	RemovePropertyDomain
Property Range	AddPropertyRange	RemovePropertyRange
Property Symmetric	AddPropertySymmetric	RemovePropertySymmetric
Property Transitive	AddPropertyTransitive	RemovePropertyTransitive
Property Inverse	AddPropertyInverse	RemovePropertyInverse
Max Cardinality	AddMaxCardinality	RemoveMaxCardinality
Min Cardinality	AddMinCardinality	RemoveMinCardinality
Instance	AddInstance	RemoveInstance
InstanceOf	AddInstanceOf	RemoveInstanceOf
InstProp	AddPropertyInstance	RemovePropertyInstance
OI-model	AddOI-model	RemoveOI-model

Table 1 – The taxonomy of ontology changes

The proposed taxonomy of changes does not include the "modify" change. Instead modify is addressed as "*rename"* and "*set"* changes:

- **Rename changes** – This change is not included in the taxonomy because it would be superfluous since there is lexical information about ontology entities (e.g. labels in different languages). To rename a concept, one has to create a new label and to remove the old one. A set of two basic changes should be executed: AddPropertyInstance and RemovePropertyInstance of label;
- **Set changes** – Again, this change is not included in the taxonomy because this type of change is a set of primitive changes. For example, setting up the inheritance relationship between two concepts (e.g. SetSubConcept change) can be achieved through two changes: the AddSubConcept change establishing the subconcept relationship between two concepts and the RemoveSubConcept change removing the subconcept relationship between concepts.

3.2.2.2 Composite Change

Composite change modifies (creates, removes or changes) the neighborhood of an ontology entity. Table 2 introduces all composite changes considered in [Stojanovic, 2004].

Composite Change	Syntax & Semantics
Pull concept up	PullConceptUp(c) Attaches a concept c to all parents of all its parents
Pull concept down	PullConceptDown(c) Attaches a concept c to the children of all its parents excluding itself
Group concepts	GroupConcepts(c1,c2, newC) Createes a common superconcept newC and transfer common properties to it
Split concept	SplitConcept(c, newC1, newC2) Splits a concept c into two concepts newC1 and newC2 and distributes properties and instances among them
Merge concepts	MergeConcepts(c1, c2, newC) Replaces concepts c1 and c2 with one concept newC and aggregates all properties and instances
Concept copy	CopyConcept (c, newC) Duplicates a concept c with all its properties and directed instances by creating a new concept newC and attaching it to all parents of c
Concept generalisation	AddConceptGeneralisation(c, newC) Adds a new concept newC between a given concept c and all its parents
Inheritance extension	AddInteriorConcept(c1,c2, newC) Adds a new concept newC and attaches it to a concept c1 as its parent and to a concept c2 as its child
Concept specialisation	AddConceptSpecialisation(c, newC) Adds a new concept newC between a given concept c and all its children

Table 2 – Composite ontology changes according to KAON approach

Notice that every one of these changes can be decomposed into a set of elementary changes, which should be executed like a transaction.

3.2.2.3 Complex Change

Complex changes encompass all “real-life” changes not included in the elementary and composite changes raising the level of abstraction and reusability even further.

Complex Change	Semantics
Move Concept	Moves concept, which is generalisation of the PullConceptUp (or PullConceptDown) composite changes since it attaches one concept to another arbitrary concept not necessary related to the first concept through the inheritance relationship;
Move sibling Concepts	Moves a set of sibling concepts to a different location, which move two or more concepts that are siblings in the concept hierarchy to the same new location in the concept hierarchy (i.e. they remain siblings, but under different parent);
Deep Concept Copy	Recursively applies “shallow“ copy (i.e. the CopyConcept composite change) to all sub concepts of a considered concept

Table 3 – Complex change examples according to KAON approach

An elementary and composite change does not include all transformation that might be desired in the future. There are always possibilities for defining a new complex change by combining existing ontology changes. While it is not possible to enumerate them, Table 3 presents some of them, developed and applied in the context of KAON.

3.2.3 Consistency Maintenance

The module responsible for keeping the ontology consistent in KAON ontology evolution approach is called semantics of change.

Application of an elementary change to ontology can induce inconsistencies in other parts of the ontology. The task of the semantics of change phase is to enable the resolution of induced changes in a systematic manner, ensuring consistency of the whole ontology.

The KAON ontology evolution approach distinguishes structural and semantic inconsistency. Structural inconsistency arises when the *constraints of the ontology model* are invalidated (e.g. undefined entities at the ontology or instance level are used). Semantic inconsistency arises when the *meaning* of an ontology entity is changed due to the changes performed in the ontology.

3.2.3.1 Semantic Inconsistency Resolution

For allowing semantic inconsistency resolution, the standard ontology model has to be enriched with semantic information that exactly characterizes the concept's semantic properties and expected ambiguities. This includes the properties that are prototypical for a concept and that are exceptional or essential. Additionally, the behavior of the properties over time and the degree of applicability of properties to sub concepts might (or should) be address too.

[Stojanovic, 2004] approach focus only in structural inconsistency resolution since the semantic inconsistency depends on additional (semantic) information added to ontology model.

3.2.3.2 Structural Inconsistency Resolution

Structural inconsistency arises when the ontology model constraints are invalidated. Since ontology engineer can easily overlook some part of the overall modification, it cannot be expected that the generation of additional changes needed to keep the consistency can be performed manually. Accordingly, two main approaches were adopted from the database community to ensure the consistency:

- **The Procedural approach** - This approach resembles classical problems of schema change that specify the semantics of ontology changes (i.e. pre-conditions and post-conditions) and accordingly, specify rules to preserve the consistency of the resulting ontology. This approach is extended by specifying a multiple set of rules (i.e. the evolution strategies) for ensuring the consistency. Additionally, the so-called meta-rules (i.e. the advanced evolution strategies) are defined for controlling the set of consistency enforcing rules that have to be used;
- **The Declarative approach** - This approach models the ontology evolution as a reconfiguration-design problem solving task, such that 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 a target one. The search of the graph is guided by the constraints provided partially by an ontology engineer, and partially by a set of rules defining the ontology consistency.

3.2.3.3 Evolution Strategies

Evolution strategy is responsible for:

- Ensuring that all changes applied to the ontology, leave the ontology in a consistent state;
- Preventing illegal changes.

Evolution strategies is a method of finding the consistent ontology that meets the needs of the ontology engineer, and by observing and analyzing the users behavior in order to anticipate his needs. Since an elementary change may cause inconsistencies in other part of the ontology, some other changes needs to be fired to keep consistency. An evolution strategy is a set of rules defined upon the management of ontology entities and their inter-relation. For example, what happens with sub concepts, ranges, domains, etc., when deleting a concept from the ontology?

3.2.4 Storing Changes

In KAON ontology evolution approach, all changes performed in ontology are stored in a log file. This is the last task of the change implementation phase. The KAON ontology evolution approach makes use of an **Evolution Ontology** and the **Evolution Log**.

3.2.4.1 Evolution ontology

The Evolution Ontology [Stojanovic, 2004] is a meta-ontology that is used as a backbone for creating Evolution Logs. The ontology was created according to concepts introduced in change representation phase (see 3.2.2). Therefore, the most important concept is the *Change* concept. The hierarchy structure of ontology changes reflects the underlying ontology model by including all possible types of changes.

The evolution ontology models what changes and their semantics, and additionally, meta-data such as why, when, by whom are performed upon an ontology. Figure 14 depicts the KAON GUI representation of a part of the Evolution Ontology, including classes of changes and their relations.

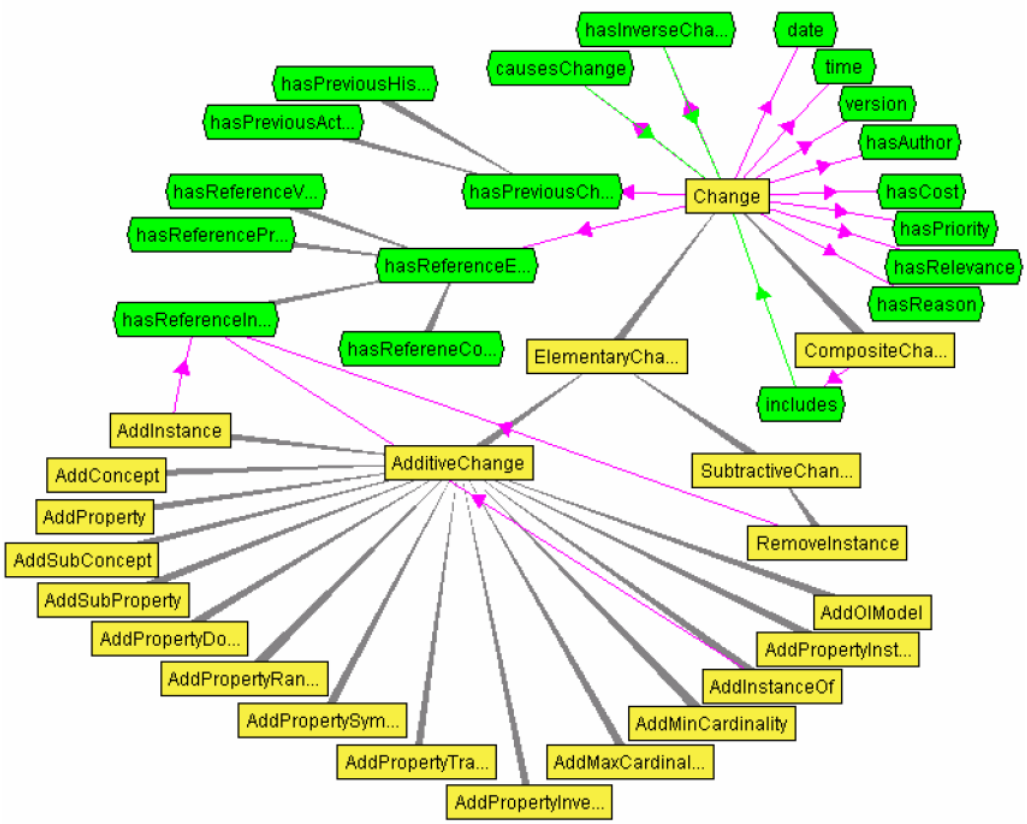

Figure 14 – Partial graphical representation of Evolution Ontology

3.2.4.2 Evolution Log

An evolution log is an instantiation of the evolution ontology. The Evolution Log records an exact sequence of changes that occurred when an ontology engineer updates the ontology. Therefore, it contains instances of sub concepts of the *Change* concept, which include the elementary as well as composite ontology changes. Each instance contains data about a particular change. Moreover, all changes in the log have a timestamp, author, version, etc. The data in evolution log can be accessed using the KAON API.

3.3 WISE Ontology Evolution Approach

This section describes the main ideas of the WISE ontology evolution approach, proposed by [Plessers, 2005].

The approach is based on keeping track of the different versions of ontology concepts throughout their lifetime (called virtual versions). In this way, changes can be defined in terms of these virtual versions.

3.3.1 WISE Ontology Evolution Overview

This approach divides evolution process in five phases depicted in Figure 15:

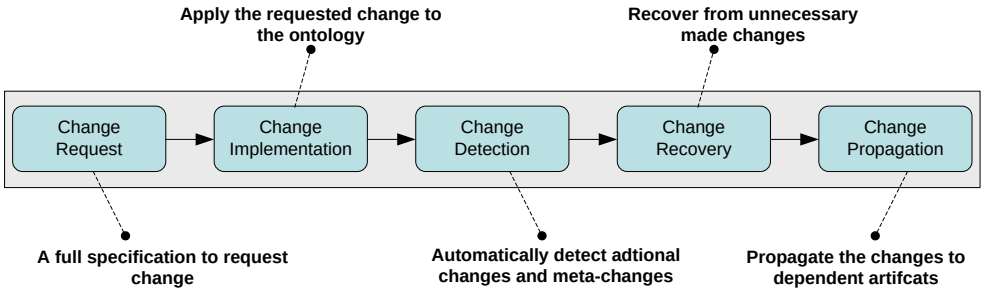

Figure 15 – Five phases of the WISE ontology evolution approach

- **Change request** – In the first step, the ontology engineer specifies the request for change in terms of basic and composite changes. In the second step, it is checked whether the ontology remains consistent if the requested change would be applied. If this is not the case, new changes (called deduced changes) are added to the change request to solve the inconsistencies;
- **Change implementation** – Takes as input the complete change request specification of the previous phase, and executes the specified changes on the ontology;
- **Change detection** – Checks if other changes or meta-changes occur as a consequence of the ontology modification;
- **Change recovery** – Deduced changes from the change request phase are checked and possibly need to be revised;
- **Change propagation** – Depending artifacts are brought into a consistent state by propagating changes listed in the evolution log.

Beyond these five phases, this approach has a key element named **Version log**. This log keeps track of the different versions (virtual versions) of the ontology concepts during its lifetime: starting from the creation of the concept, over its modifications, until its retirement. A version log does not specify what changes, it only list the successive versions. The version log is an instantiation of the version ontology (Figure 16).

For each class, property and individual that are created in the ontology, an associate Evolution Concept instance from the version ontology is created in the version log. Such an Evolution Concept instance keeps, besides a reference to the concept in the ontology, a list of past and current versions of the referred concept.

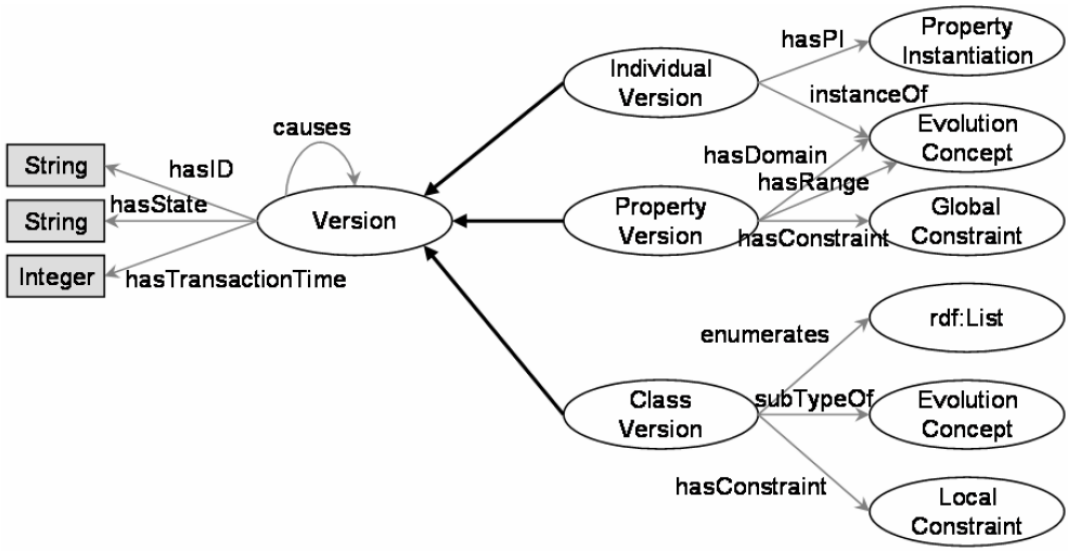

Figure 16 – The Version concepts of the version ontology

Whenever a change request for a concept in the ontology is executed, a new Version instance is added to the associated Evolution Concept instance, representing the new version of the referred concept.

3.3.2 Change Representation

In this section the WISE ontology evolution approach and its ontology changes representation are described. According to [Plessers, 2005] an ontology change is an interpretation of an ontology modification. The definition of a change formally specifies the modifications that correspond with this change.

The Change Request phase gives the ontology engineers the ability to specify their request for change through predefined changes. Different types of changes are distinguished by the ontology evolution framework:

- **Primitive Changes** – Modify exactly one element of the ontology and cannot be decomposed any further. Primitive Changes are in turn divided into domain independent and domain dependent changes. The set of domain dependent changes are defined for a particular domain. The set of domain independent changes is derived from the underlying ontology language and expressed as a set of modifications to the constructs of the ontology language;
- **Composite Changes** – This type of changes modify more than one element of the ontology and offer a higher level of abstraction;
- **Meta Changes** – This type of changes specify the implications of a change. They provide information about a change.

3.3.3 Consistency Maintenance

Consistency maintenance is address in the WISE approach through the concept of "deduced change". The process checks if the ontology remains consistent if the requested change would be applied. If not, new changes (deduced changes) are added to the change request for solving inconsistencies. This process is iterative since deduced changes may result in additional deduced changes.

To check for consistency, a consistency model is used. Such consistency model is a formal meta-model that restricts the version ontology so that it is conforms to the constraints imposed by the ontology language used. This means that whenever the latest version stored in the version log conforms to the consistency model, the requested changes can be applied. Notice that different consistency models may exist for different ontology languages or different variants of an ontology language (e.g. OWL Lite and OWL DL).

The result of the **Change Request** phase is a complete **change request specification** composed of requested and deduced changes that transform an ontology from one consistent version into another consistent version.

In the **Change Recovery** phase, the deduced changes from the Change Request phase are checked and possibly need to be revised. Some deduced changes may be not necessary after an execution of a set of requested changes. Ontology engineer may have used a sequence of basic changes instead of the right complex change. New changes will be deduced after each basic change to keep the ontology consistent, but after all, these deduced changes may be superfluous. This superfluous deduced changes need to be undoing.

3.3.4 Storing Changes

This approach uses the concept of evolution log, in which the changes listed in the change request are added. This log keeps track of all the applied changes and gives an overview of the complete evolution history of the ontology in respect to changes.

3.4 SIKS Ontology Evolution Approach

In this section the basic ideas of SIKS[7] Ontology Evolution Approach [Klein, 2004] are presented.. Klein's approach is quite different from the two previous. An ontology change is defined by [Klein, 2004] as an action on ontology that results in an ontology that is different from the original version. This work, entitled "*Change Management for Distributed Ontologies",* assumes that there are online, distributed ontologies without a central authority that publishes or authorizes ontology version. It assumes that:

- there is a world (e.g. the Semantic Web) where many versions of ontologies can exist;

[7] *The Dutch Research School for Information and Knowledge [http://www.siks.nl]*

- all of these versions may still be in use;
- ontologies will evolve in unpredictable way.

For that reason, any version can be useful depending on the task one wants to perform.

3.4.1 SIKS Ontology Evolution Overview

[Klein, 2004] defines three levels of changes based on the definition of ontology presented by[Gruber, 1993], "**Specification** of a **conceptualization"**, which is usually **represented** in some ontology language:

- **Conceptual change** – Respects the change in the conceptualization. In other words, it is a change in the way a domain is interpreted , which results in different ontological concepts or different relations between these concepts;
- **Specification change** – Change in the specification of a conceptualization. In other words, the way the conceptualization is defined, without changing the conceptualization itself;
- **Representation change** – Change in the representation of a specification of a conceptualization. Typically a change at the syntactic (language) level.

A change in the specification does not necessarily coincide with a change in the conceptualization, and changes in the specification of ontology are not per definition ontological changes.

Based in previous assumptions and foundations, [Klein, 2004] presents the ontology change process model (Figure 17).

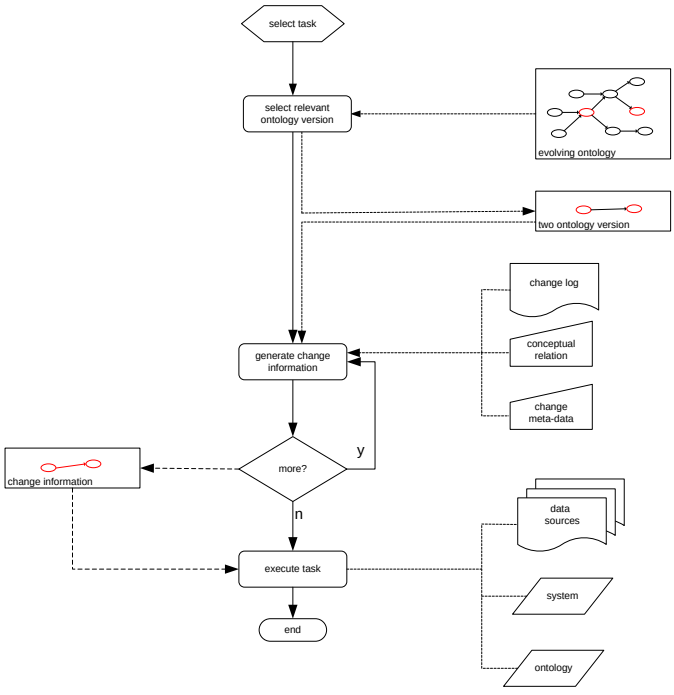

Figure 17 – The process of managing ontology change

This process comprehends two main steps:

- **Determine the specific versions** – Starting from a selected task (i.e. data retrieval), the goal is to determine the versions from the ontology life trace that are relevant for that task at hand. The selection procedure is different for the each task. For example, for change validation and approval (i.e a task were a person has to approve ontology changes that are proposed) it usually means that the newly created version and the original version are selected. Another example, in data retrieval, the version that is used to annotate the data and the version that will be used to retrieve the data should be selected. Evolution logs and structural diff provided by other tools can be used to perform the selection. The result of this step is the identification of two ontology versions;
- **Generating the change specification** - This step consists of generating the change specification between these two versions. Dependent on the available information and the required knowledge for the tasks, new knowledge about the relation between the two identified ontology versions will be derived. When enough information about the change between the identified versions is available, it can be used to perform the selected task, for example on mapping data sources. If, and to what extent the tasks can be performed successfully, depends on the knowledge about the change that is available in the end of this phase. In general, the aim of the change management methodology is allowing tasks to be performed at least partially.

In next sections present how ontology changes are represented (section 3.4.2), how ontology consistency is kept (section 3.3.3) and how ontology changes are stored (section 3.3.4).

3.4.2 Change Representation

According to [Klein, 2004], changes between two ontology versions can be represented in a number of different ways. Figure 18 illustrates the different change representations used in SIKS ontology evolution approach and their transformations.

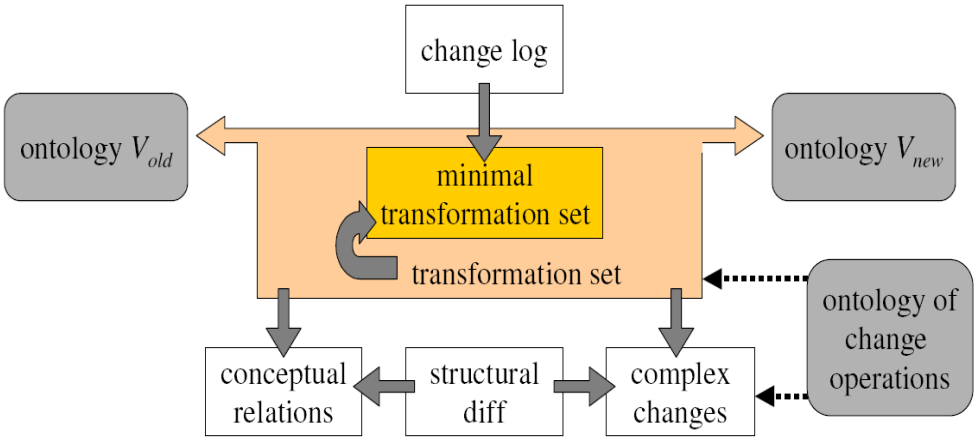

Figure 18 – A schematic representation of relations between different change representations.

Starting with two versions V_{old} and V_{new} of an ontology, there are the following forms of representation:

- **Ontologies only** – The old and new ontology version provide no explicit change information, but can be used as starting point to find other change information;
- **Change Log** – Record of changes as they are performed, i.e. sequential changes that were applied to V_{old} to generate V_{new};
- **Structural diff** – A mapping between entities of old ontology version and their counterparts in the new ontology version, together with a list of added and removed concepts;
- **Conceptual relation** – Explicit specification of the conceptual relations between concepts (equivalence relations) in V_{old} and corresponding concepts in $V_{new.}$

Based on these representation requirements, the following main components of SIKS ontology evolution approach are defined:

- **Meta ontology of change operations** – Specifies a large number of standard changes to an ontology;
- **Complex changes** – Is a set of named groups of basic change operations. Each group constitute a logical change entity;
- **Transformation set** – Provides a set of operations that specify how V_{new} can be transformed in V_{old};
- **Template for change specification** – Describes how two ontology versions are related.

Next four sub-sections describe each of these components.

3.4.2.1 Meta Ontology of Change Operations

Ontology of change operation specifies a large number of standard changes to an ontology. These operations are directly related to the ontology language (*e.g.* OWL). [Klein, 2004] defined ontology change operations for OWL language and OKBC[8] according to the correspondent meta-model. Table 4 presents a summary of the change operations for entities specified in the OWL/RDF language.

Object	Operation(s)	Argument(s)
Class	Add, Remove	Class definition
Restriction	Add, Remove	Restriction definition + type
Restriction filler	Modify	2 Restriction fillers
Cardinality upper/lowerbound	Add, Remove	Cardinality restriction
Cardinality upper/lowerbound	Modify	Property ID + 2 values
Restriction type	Make ∀ make ∃	Restriction definition
Super class relation	Add, Remove, Modify	1 or 2 class definitions

[8] *Open Knowledge Base Connectivity (http://en.wikipedia.org/wiki/Open_Knowledge_Base_Connectivity)*

Class disjointness	Add, Remove, Modify	1 or 2 class definitions
Class equivalence	Add, Remove, Modify	1 or 2 class definitions
Property	Add, Remove	Property definition
Domain	Add, Remove, Modify	1 or 2 Property definitions
Range	Add, Remove, Modify	1 or 2 Property definitions
Resource type	Add, Remove, Modify	1or 2 class ID
Resource label	Add, Remove, Modify	1 or 2 strings

Table 4 – Examples of change operations for OWL languages

The first column represents the object where operation is applied; the second represents the applied operation and the third, the arguments. The number of argument depends on the chosen operation. The Modify operation specifies that an old value is replaced by a new value. Notice that, despite the old value is not necessary to execute the Modify operation, it is useful for reversing the operation.

3.4.2.2 Complex Changes

This provides the mechanism for grouping a number of basic operations that together constitute a logical change entity. Unlike each of the basic change operation defined in previews section that modifies only one specific feature in the model, complex change modifies more than one single feature. Table 5 illustrates an excerpt of the complex changes proposed in SIKS ontology evolution approach.

Operation	Type
Restrict_Range	R
Modify_Range_To_Subcass	R
Extend_Range	R
Modify_Range_To_Superclass	R
Restric_Domain	R
Modify_Domain_To_Subcass	R
Extend_Domain	R
Modify_Domain_To_Superclass	R
Add_Subtree	R
Delete_Subtree	C
Move_Subtree	C
Delete_Subclass	C
Restrict_Cardinality	CR
Extend_Cardinality	CR
Change_To_Primitive	CR
Change_To_Difined	CR

Table 5 – Example of complex change operations for OWL

The first column represents the change operation and the second its type, corresponding to:

- The R type (**Rich** operation), incorporates information about the implication of the operation on the logical model of the ontology;
- The C type (**Composite** operation), provides a mechanism for grouping a number of basic operations together constituting a logical entity.

3.4.2.3 Transformation Set

A transformation set provides a set of change operations instances that specify how V_{old} ontology can be transformed in V_{new} ontology. A transformation set is not unique; there are often multiple ways to construct a transformation set for a specific change.

Figure 19 depicts an example of a transformation set, in which every change includes two arguments. The "addSuperclass" change states that the "Cabernet blanc" class is super class of the "Rosé wine" class.

```
changeName oldName=Riesling, newName=Weisser Riesling
addSuperclass child=Cabernet blanc, parent=Rosé wine
addSuperclass child=Vin gris, parent=Rosé wine
removeSuperclass child=Cabernet blanc, parent=white wine
removeSuperclass child=Vin gris, parent=White wine
```

Figure 19 – Fragment of a transformation set in SIKS ontology evolution approach

3.4.2.4 Template for Change Specification

A template is used for describing how two ontology versions are related. A template has the following elements:

- **Descriptive meta-data** – Book-keeping information like **date** of release, **author** of the changes, and the number of the versions. This describes the what, when and the who of the change;
- **Minimal transformation set** – The kernel of the template, as it forms a complete operational specification of the change. It can be used to re-execute the change, to translate or re-interpret data sets, and as a basis for deriving additional information about the change;
- **Conceptual relations** – The relation between concepts across versions as specified by the ontology engineer. This facilitates data access by improving the interpretation and querying of data sources that were described with different versions of ontologies;
- **Complex changes** – Higher-level description of some of the changes. Together with the minimal transformation set, complex operations can be used to create data transformation scripts that translate/transform instances of one version of the ontology to instances of the other version. Also, complex changes allow determining in more detail the effect of changes on data accessibility and specific logical queries;
- **Change rationale** – The intention behind the change. The rationale specifies whether the change is a fix of an error, a more specific description, or an update of the real world. The intention can be used to decide which version to use and can help to visualize the change.

3.4.3 Consistency Maintenance

In this approach, there are no changes to ontology that make it inconsistent, but instead, the consistency depends on the task at hand.

In fact, this approach assumes that there are many versions of the same ontology, and they are selected according to the task and context.

3.4.4 Storing Changes

According to [Klein, 2004], changes between two ontology versions can be represented in a number of different ways: Ontologies only, Log of changes, Structural diff, Conceptual relation and additionally, a transformation set. Yet, because each form has different features and goals, [Klein, 2004] defines a set of procedures for transforming between change representation forms:

- **V_{old} and V_{new}, → transformation set** – With both V_{old} and V_{new}, the two versions can be compared to create automatically a transformation set between them;
- **V_{old} and V_{new}, → structural diff –** Starting from V_{old} and V_{new}, a structural diff between both can also be created. This specification maps the concepts and properties in the old version to their counterpart in the new version;
- **V_{old}, V_{new} and structural diff → conceptual relations** – When both versions of the ontology and the evolution relation between concepts are available, an automated reasoner can be used to derive conceptual relations between the concepts in the old and new version of the ontology**;**
- **Structural diff → complex changes** – Structural diff can be used to create more useful change descriptions;
- **Structural diff → conceptual relations** – Conceptual relations can be derived from a structural diff. For instance, the mappings of a structural diff directly suggest equivalence relations between concepts;
- **Change log → minimal transformation set** – Log provides by many ontology-editing tools can be transformed into transformation sets by translating the operations into the vocabulary of basic changes and removing redundant changes;
- **Transformation set → minimal transformation set** – Transformation set can be transformed into minimal transformation sets by removing all redundant changes;
- **Transformation set → complex changes** – Transformation set consisting of basic operations, can be extended with heuristics to combine the basic operations into complex change operations;
- **Transformation set → conceptual relations** – Transformation set with both basic and complex operations defined between versions, can be extended by heuristics to suggest conceptual relations between frames in versions to the user.

3.5 Conclusion

The goal of this chapter has been the identification and comprehension of three dimensions of the ontology evolution process:

- What and how ontology changes are represented;
- How ontology changes are processed to keep ontology consistent;
- How ontology changes are stored and how can they be accessed.

Next sections present a systematization and comparison of the three projects/approaches described, envisaging possible applications of the analyzed approaches.

3.5.1 How Ontology Changes are Represented?

After analyzing the ontology evolution approaches proposed by[Stojanovic, 2004], [Klein, 2004] and [Plessers, 2005], it is possible to conclude that the set of ontology changes are very similar. All approaches have at least two levels of changes abstraction (basic and composite) and the notion of derived/deduced changes:

- **Basic changes** – Changes one entity in ontology model only;
- **Composite changes** – Changes more than one entity in ontology model. Can be considered as a group of basic changes;
- **Derived changes** – Since basic/composite changes can cause inconsistencies in ontology model, new changes need to be done to solve those inconsistencies.

These concepts can be useful to our work, since it is an evolution process too. Ontology mapping changes need to be represented in a formal way, and define different levels of granularity can be positive to the ontology mapping evolution process.

Besides these, [Klein, 2004] identifies, according to ontology definition presented by [Gruber, 1993], three levels of changes (i.e. changes in conceptualization, changes in specification and changes in representation). Usually, ontology evolution systems deal with changes in specification. Because ontology mapping process is a very semantic oriented, taking into account changes in conceptualization can be positive to ontology mapping evolution process.

3.5.2 How Changes are Processed to Keep Ontology Consistent?

The execution of a single ontology change (e.g. remove a concept) may cause inconsistencies in other parts of the ontology. For solving these inconsistencies, the analyzed approaches use the concept of derived or deduced changes, but because deduced changes may result in additional deduced changes, the process is iterative.

Distinguishing between changes requested by the ontology engineer (representing their intentions) and the deduced changes (to solve inconsistencies) is useful information to use in ontology mapping evolution process e.g. the relation between changes (which changes fires which changes).

[Stojanovic, 2004] presents a concept of ontology evolution strategy as a method of finding the consistent ontology that meets the needs of the ontology engineer.. This concept can be potentially useful to ontology mapping evolution process.

3.5.3 How Ontology Changes are Stored?

The approaches mentioned in previous sections, as well as others ontology development tools like Protégé[9], present a concept of evolution log. An evolution log keeps track of all ontology changes.

In [Stojanovic 2004] the evolution log is an instantiation of the evolution ontology. There is an API (KAON API) provide by KAON ontology evolution approach that can be very useful for this work since it allows reading the evolution log and indentify all ontology changes.

Accordingly, the knowledge acquired in this chapter will be useful both as research starting point for the next chapters, and as input to the envisaged ontology mapping evolution process.

[9] *http://protege.stanford.edu/*

Chapter 4

Ontology Mapping Evolution Process

An ontology mapping document is a document where the semantic relation between source ontology entities and target ontology entities are specified, therefore if ontology entities change, the relation between them (i.e. the ontology mapping document) might change too.

This chapter presents the proposed approach for ontology mapping evolution process. Understanding the last two chapters is very important for this one.

4.1 Mapping Evolution Changes

The set of mapping evolution changes depends on the ontology mapping language. The change operations have to be defined according to a set of entities of ontology mapping language. Besides these changes, another important issue is the relation between these changes. In other words, what changes may induce other changes.

Based on SBO entities, a set of ontology mapping changes, a dependency matrix between these changes and a set of composite mapping changes are defined in the following sub-sections.

4.1.1 Set of Ontology Mapping Changes

Similar to other evolution approaches such as object-oriented schema evolution proposed in [Banerjee, et al., 1987] and ontology evolution (e.g. [Klein, 2004] and [Stojanovic, 2004]) approaches, and according to the SBO language, a set of ontology mapping changes is defined. Table 6 presents a set of ontology mapping changes for SBO language. Mapping changes are represented as subtractive and additive changes as is illustrated in second and third column respectively. These changes are considered the elementary ontology mapping evolution changes, in the sense that they just change one entity of SBO ontology model.

SBO Entities	Subtractive Changes	Additive Changes
Concept Bridge	Remove Concept Bridge (**R_*CB***)	Add Concept Bridge (**A_*CB***)
Concept Bridge Source Concept	Remove Concept Bridge Source Concept Value (**R_*CB*_SCV**)	Add Concept Bridge Source Concept Value (**A_*CB*_SCV**)
Concept Bridge Target Concept	Remove Concept Bridge Target Concept Value (**R_*CB*_TCV**)	Add Concept Bridge Source Concept Value (**A_*CB*_TCV**)
Concept Bridge Generic Condition	Remove Concept Bridge Generic Condition (**R_*CB*_GC**)	Add Concept Bridge Generic Condition (**A_*CB*_GC**)
Concept Bridge Extensional Specification	Remove Concept Bridge Extensional Specification (**R_*CB*_ES**)	Add Concept Bridge Extensional Specification (**A_*CB*_ES**)
SubBridgeOf Relation	Remove SubBridgeOf (**R_SB**)	Add SubBridgeOf (**A_SB**)
HasBridge Relation	Remove HasBridge (**R_HB**)	Add HasBridge (**A_HB**)
Property Bridge	Remove Property Bridge (**R_PB**)	Add Property Bridge (**A_PB**)
Property Bridge Source Argument	Remove Property Bridge Source Argument Value (**R_PB_SAV**)	Add Property Bridge Source Argument Value (**A_PB_SAV**)
Property Bridge Target Argument	Remove Property Bridge Target Argument Value (**R_PB_TAV**)	Add Property Bridge Target Argument Value (**A_PB_TAV**)
Property Bridge Generic Condition	Remove Property Generic Condition (**R_PB_GC**)	Remove Property Generic Condition (**A_PB_GC**)
Property Bridge Service	Remove Property Bridge Service (**R_PB_S**)	Add Property Bridge Service (**A_PB_S**)
Alternative Bridge of Property Bridges	Remove Alternative Property Bridge (**R_AB_PB**)	Add Alternative Property Bridge (**A_AB_PB**)
Alternative Bridge of Concept Bridges	Remove Alternative Concept Bridge (**R_AB_*CB***)	Add Alternative Concept Bridge (**A_AB_*CB***)

Table 6 – Set of elementary ontology mapping changes

4.1.2 Ontology Mapping Changes Dependency Matrix

Similar to others evolution approaches ([Stojanovic, 2004], [Banerjee, et al., 1987]), changes are related. The execution of a single mapping change may cause inconsistencies in other parts of ontology mapping document. For solving such inconsistencies, some others changes are performed as consequence of the first one. There is a cause-effect relation (dependency) between ontology mapping changes, where for each change, a set of consequent changes exist. Table 7 represents the cause-effect relation between the elementary ontology mapping evolution changes referred in previous section. If the consequent changes of an executed change are not performed, the mapping document becomes inconsistent, therefore invalid in many contexts such as data translation.

	R_*CB*	R_*CB*_SCV	R_*CB*_TCV	R_*CB*_GC	R_*CB*_ES	R_SB	R_HB	R_PB	R_PB_SAV	R_PB_TAV	R_PB_GC	R_PB_S	R_AB_PB	R_AB_*CB*	A_*CB*	A_*CB*_SCV	A_*CB*_TCV	A_*CB*_GC	A_*CB*_ES	A_SB	A_HB	A_PB	A_PB_SAV	A_PB_TAV	A_PB_GC	A_PB_S	A_AB_PB	A_AB_*CB*
R_*CB*	-	X	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R_*CB*_SCV	X?	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X?	-	-	--	-	-	-	-	-	-	-	-	-
R_*CB*_TCV	X?	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R_*CB*_GC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X?	-	-	-	-	-	-	-	-	-	-
R_*CB*_ES	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R_SB	?	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R_HB	-	-	-	-	-	-		?	-	-	-	-	?	?	-	-	-	-	-	?	-	-	-	-	-	-	-	-
R_PB	-	-	-	-	-	-	X	-	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R_PB_SAV	-	-	-	-	-	-	-	X?	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X?	-	-	-	-	-
R_PB_TAV	-	-	-	-	-	-	-	X?	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X?	-	-	-	-
R_PB_GC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R_PB_S	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R_AB_PB	-	-	-	-		-	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
R_AB_*CB*	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_*CB*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	-	-	-	-	-	-	-	-	-	-	-
A_*CB*_SCV	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_*CB*_TCV	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_*CB*_GC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_*CB*_ES	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_SB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_HB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_PB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	-	X	X	-	X	-	-
A_PB_SAV	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_PB_TAV	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_PB_GC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_PB_S	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A_AB_PB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-
A_AB_*CB*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-

Table 7 – Dependency matrix of ontology mapping changes

- **X** – The value **X** of an element, i.e. Dependency[i][j]=**X**, indicates that the resolution of a change related to the row i induces a change related to the column j. For example, the ontology mapping change **R_*CB*** (Remove Concept Bridge) induces the following ontology mapping changes: **R_*CB*_SCV** (Remove Concept Bridge Source Argument Value)**, R_*CB*_TCV** (Remove Concept Bridge Target Argument Value)**, R_*CB*_GC** (Remove Concept Bridge Generic Condition)**, R_*CB*_ES** (Remove Concept Bridge Extensional Specification)**, R_SB** (Remove Concept Bridge Sub Bridge of), **R_HB** (Remove Has Bridge);
- **?** - The value **?** of an element, i.e. Dependency[i][j]=**?**, indicates that the resolution of a change related to the row i **might** induce a change related to the column j. For example, the change **R_SB** (Remove Sub Bridge of) might induce **R_*CB*** (Remove Concept Bridge) change;
- **X?** – The value **X?** of an element, i.e. Dependency[i][j]=**X?**, indicates that the resolution of a change related to the row i induces **one of** changes related to the column j. For example, the change **R_*CB*_SCV** (Remove Concept Bridge Source Concept Value) induces **R_*CB*** (Remove Concept Bridge) change **OR A_*CB*_SCV** (Add Concept Bridge Source Concept Value) change.

4.1.3 Composite Mapping Changes

The set of ontology mapping changes (Table 6) and the dependency matrix (Table 7), promotes the concept of **composite mapping change**. Composite mapping changes can be defined as the combination of two or more elementary mapping changes organized as a logical unit that should be executed as a whole. Table 8 presents a set of useful composite mapping changes, used during this work.

Composite Mapping Change	Elementary Mapping Changes
Change Concept Bridge Source Concept Value	Remove Concept Bridge Source Concept Value + Add Concept Bridge Source Concept Value
Change Concept Bridge Target Concept Value	Remove Concept Bridge Target Concept Value + Add Concept Bridge Target Concept Value
Change Concept Bridge Generic Condition	Remove Concept Bridge Generic Condition + Add Concept Bridge Generic Condition
Change Concept Bridge Extensional Specification	Remove Concept Bridge Extensional Specification + Add Concept Bridge Extensional Specification
Change Property Bridge Has Bridge	Remove Property Bridge Has Bridge + Add Property Bridge Has Bridge
Change Property Bridge Source Argument Value	Remove Property Bridge Source Argument Value + Add Property Bridge Source Argument Value
Change Property Bridge Target Argument Value	Remove Property Bridge Target Argument Value + Add Property Bridge Target Argument Value
Change Property Bridge Generic Condition	Remove Property Bridge Generic Condition + Add Property Bridge Generic Condition
Change Property Bridge Service	Remove Property Bridge Service + Add Property Bridge Service

Table 8 – Composite mapping changes

Composite mapping changes was defined by observing the dependency matrix of ontology mapping changes one can notice cells containing "X?" value. The **X?** value means that for one row (representing a change), there are a set of mutually exclusive consequent changes available. Based on Dependency[i][j]=**X?**, a set of composite mapping changes has been identified. However, there are many composite mapping changes that can be created since it is a combination of elementary mapping changes.

The notion of composite mapping change allows:

- Higher the level of abstraction, as it allows the specification of coarse-grained;
- Represents a semantic context of the mapping evolution process, because it groups together a set of meaningful and complementary elementary changes, representing a semantic change;
- Provides additional semantic meaning to the change, since it nominates (gives a name to) a set of meaningful, complementary changes according to their overall grouped goal/achievement, and not by each individual, independent goal.

4.2 Mapping Evolution Process Overview

Ontology mapping evolution process is a process whereby entities of ontology mapping document is adapted regarding eventual changes in source and/or target ontologies, trying to preserve as much as possible the semantics of ontology mapping semantic relations.

If changes occurs on source/target ontology (see Chapter 3), some semantic relations in ontology mapping document may become invalid and/or new semantic relations may be necessary for new ontology entities. For that, the ontology mapping evolution process has been developed. This process comprises two sub processes:

- Correct Invalid SBO entities;
- Process new ontology entities.

Moreover, the mapping evolution process is an iterative process. In fact, because processing new entities might invalidate the previous processed ones, the process iteratively revise their consistency, resulting in the iterative process (Figure 20).

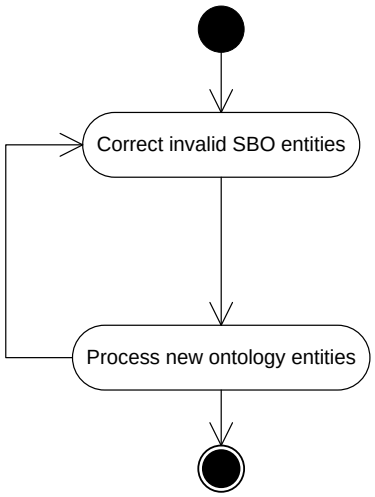

Figure 20 – Mapping evolution process overview

Processing new ontology entities requires discovering and similarity calculation between entities and specification of semantic relations according to the SBO language. MAFRA Toolkit's support to (semi) automatic ontology mapping process is described in [Silva, 2004]. Ontology mapping process is not the goal of this work, so the processing of the new ontology entities was not consider, but an overview of the implemented process in MAFRA Toolkit is presented in section 4.4.

Next section describes the process of correcting invalid SBO entities.

4.3 Correct Invalid SBO Entities

Some SBO entities may become invalid if source and/or target ontologies changes. This process is responsible for (i) get the invalid SBO entities, (ii) the reason why they are invalid and (iii) correct them. Information provided by ontology evolution phase is used in this process. After this process, the ontology mapping document should be structurally consistent.

Figure 21 is an overview of the process correcting invalid SBO Entities (Concept Bridges, Property Bridges and Alternative Bridges).

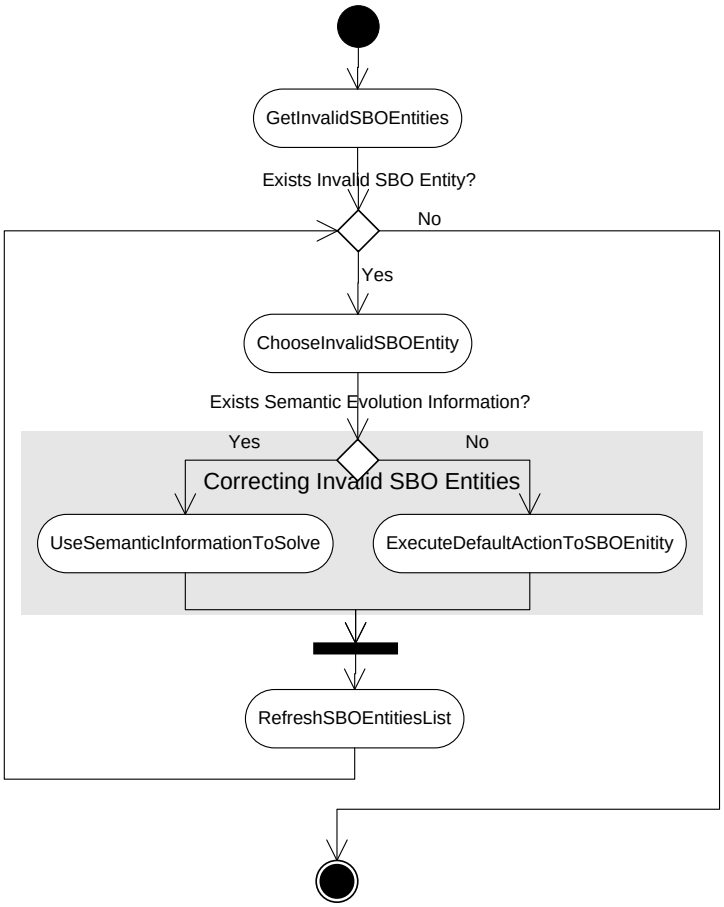

Figure 21 – Correct invalid SBO entities overview.

The following sections describe each step of these processes.

4.3.1 Get Invalid SBO Entities List

This process is responsible for discovering and retrieving invalid SBO Entities. Yet, first it is necessary to define the characteristics of an invalid SBO Entity. In other words, what makes an SBO entity invalid?

According to [Silva, 2004], there are four SBO Entities: Concept Bridge, Property Bridge, Alternative Bridge of Property Bridge and Alternative Bridge of Concept Bridge, which have different validations restrictions.

4.3.1.1 Invalid Concept Bridge

An invalid Concept Bridge is one that has any of its argument invalid:

- **Source Concept Value**, is invalid when the entity in its concept value either:
  - does not exist anymore;
  - is not a concept;

- is not a concept from source ontology.
- **Target Concept Value**, is invalid when the entity in its concept value either:
  - does not exist anymore;
  - is not a concept;
  - is not a concept for target ontology.
- **Extensional specification** is invalid when one condition of its condition list is invalid. This work deals only with conditions evolving ontology entities, so the condition will be invalid when ontology entities change;
- **Generic Conditions** – Invalid when one condition of its condition list is invalid. This work deals only with conditions evolving ontology entities, so the condition will be invalid when ontology entities change.

4.3.1.2 Invalid Property Bridge

An invalid Property Bridge is one that at least, one of its arguments is invalid according to the applied service:

- **Source Argument Value**. This argument value becomes invalid if at least one of its argument values is invalid according to the source argument specified by the service applied to the Property Bridge;
- **Target Argument Value**. Invalid conditions are equivalent to those presented to the source argument;
- **Extensional specification**. Invalid conditions are equivalent to those presented in the Concept Bridge for the same argument;
- **Generic Conditions**. Invalid conditions are equivalent to those presented in the Concept Bridge for the same argument.

4.3.1.3 Invalid Alternative Bridges

An invalid Alternative Bridge is one that has no link to any Property Bridge or Concept Bridge. Besides Alternative Bridges may be invalid, they may be unnecessary too. In fact, because Alternative Bridge is responsible for the specification of disjoint relations between Semantic Bridges, if there is just one semantic bridge linked, the Alternative Bridge is unnecessary.

4.3.2 Choose Invalid SBO Entity

Once the list of invalid SBO entities is defined, it is necessary to select the one for correcting. This process is responsible for choosing one of the invalid SBO Entity for correcting. The order of the entity for processing is very important because there are relations between entities in that list. In fact, considering that

entities are invalid because of others and correcting one may correct others, choosing the one that minimizes changes is very important.

Contrary to RDFS ontology language which is property centric, the Semantic Bridge Ontology (SBO) Property Bridge is always defined in a context of a Concept Bridge. For that reason Concept Bridges are corrected first. Yet, because in some cases correcting a Concept Bridge often implies updating their Property Bridges (e.g. changing a Concept Bridge concept value, it is necessary to update all its Property Bridge paths. Figure 22 represents the order of invalid SBO entities for processing:

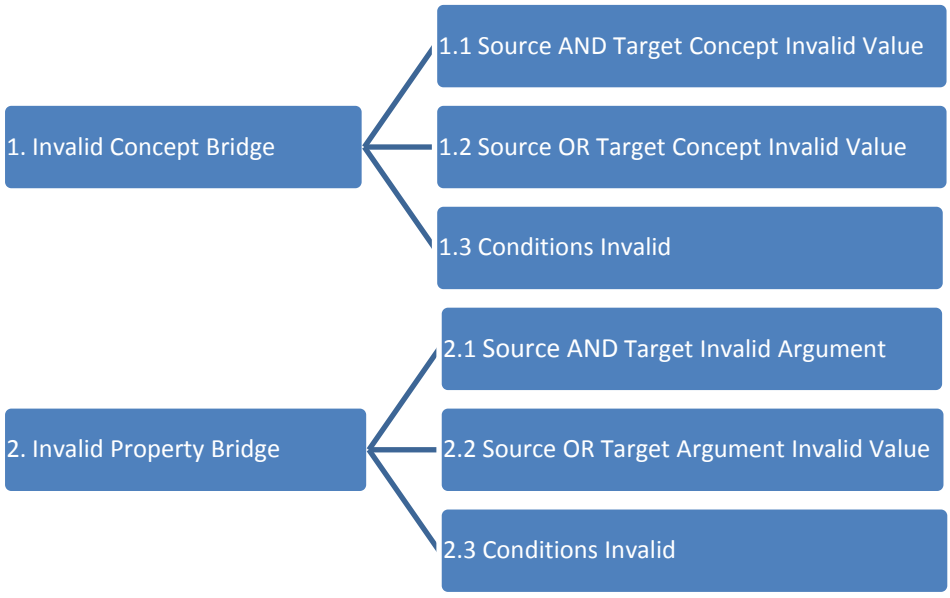

Figure 22 –Order of invalid SBO entities to process

1. **Invalid Concept Bridge** – Since in SBO language all Property Bridges are defined in a context of Concept Bridges, first, Concept Bridges are corrected and late update its Property Bridges;
   1.1. **Source AND Target Concept Invalid Value** – The firsts Concept Bridges to correct are those that have invalid source AND target concept value. If there are no source and target concepts, the only solution is removing the Concept Bridge. Choosing these Concept Bridges at first, avoid unnecessary computation;
   1.2. **Source OR Target Concept Invalid Value** – After previous step, there is no invalid Concept Bridges with invalid source AND target concept invalid. After that, invalid Concept Bridges with invalid source OR target concept value are selected for processing;
   1.3. **Invalid Conditions** – Further, after all Concept Bridges are corrected respecting source and target concept values, their conditions (extensional specification and generic conditions) are corrected.
2. **Invalid Property Bridge** – After correcting all Concept Bridges and update all Property Bridges, invalid Property Bridges are next:

2.1. **Source AND Target Invalid Argument** – First the Property Bridges with invalid source AND target argument are resolved. Removing the invalid Property Bridge may be the solution for this case, and if it happens, no further reasons should be investigated;

2.2. **Source OR Target Invalid Argument** – After correcting all Property Bridges with invalid source AND target arguments, Property Bridges with source or target argument invalid are corrected;

2.3. **Conditions Invalid** – The last step is to correct Property Bridges with invalid conditions.

4.3.3 Correct Invalid SBO Entity – using ontology evolution information

This process is responsible for exploiting ontology evolution information provided by ontology evolution phase in order to correct invalid SBO entities during ontology mapping evolution process. According to our study about ontology evolution process, usually ontology changes are stored in evolution logs provided by many tools. These logs track all ontology changes, translating the user requests during ontology evolution. The user requests during ontology evolution phase are useful during ontology mapping evolution process.

The following characteristics are examples of important and useful semantic information for ontology mapping evolution process:

- Changes are stored sequentially;
- Changes are related (cause-effect changes);
- Differences between requested and derived changes;
- Possibility to understand/capture user desires/intentions;
- Changes in meta-information are stored (e.g. label and comments).

The goal is to use information such as ontology evolution scenarios, composite changes, etc. The idea is to address ontology evolution process not like a group of atomic changes, but like a set of grouped changes as logical entity, that better captures the ontology engineer intentions.

This process is further described in Chapter 5.

4.3.4 Correct Invalid SBO Entity – no ontology evolution information

This process is responsible for correcting invalid SBO entity without ontology evolution. The ideal scenario is to have information about all user changes during ontology evolution process and their semantics in order to correct the invalid SBO entities accordingly to these changes. However, in real world, information that can be used to directly or indirectly access the ontology changes and semantics such as:

- ontology evolution logs;
- previous ontology version;
- transformation sets

is rarely available.

Nevertheless, the correction of invalid SBO entities needs to be done. This process is further described in Chapter 6.

4.3.5 Refreshing Invalid SBO Entity List

This process is responsible for refreshing the list of invalid SBO Entities. Since the process of correcting an invalid SBO entity (using or not ontology evolution information) might correct others, this process updates the list, removing the already valid entities.

4.4 Process New Ontology Entities

After correcting all invalid SBO entities, the ontology mapping document should be structurally valid. Beside this, in ontology evolution process new ontology entities may be added to ontology. Probably new semantic relations should be specified for these new ontology entities. The process whereby semantic relations are defined between two ontologies at the conceptual level is called ontology mapping.

Despite this is not the main purpose of this work, this section presents an overview of the (semi) automatic ontology mapping process in MAFRA Toolkit, presenting the most important steps: (i) Similarity measure and (i) automatic semantic bridging. These processes are responsible for discovering the similarity between ontology entities and establishing the (correct) semantic relations between them. The result is the ontology mapping document as an instantiation of SBO (Semantic Bridge Ontology). Figure 23 illustrates the overview of the (semi) automatic ontology mapping process according to MAFRA Toolkit architecture.

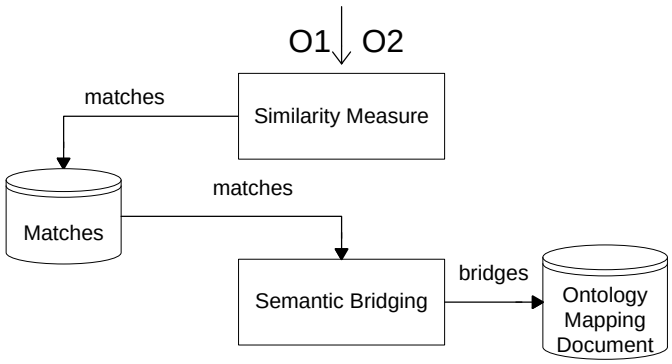

Figure 23 – Overview of the (semi) automatic ontology mapping process

4.4.1 Similarity Measure

The Similarity Measure phase aims to discover and measure similarities between source ontology's entities and target ontology's entities. The task associated with this process is also known as matching [Euzenat, et al., 2007],[Rahm, et al., 2001].

In MAFRA Toolkit context, this process is performed by matchers (Matching Algorithm Systems), which evaluate the similarity between pairs of source and target ontology's entities. The similarity of a pair of entities is referred as Match (Correspondence in ontology matching terminology) and denotes the confidence that a specific Matcher has about the similarity relation between source and target ontology's entities. In context of MAFRA Toolkit, a Match is a structure in form of:

$$m = \{e, e', M, r, c\}$$

Where:

- *e* and *e'* are respectively the source and the target ontology's entities;
- *M* is the Matcher that evaluated the match;
- *r* is the relation holding between *e* and *e'*. Typically are '<', '=' and '>';
- *c* is the confidence value assigned to the similarity between *e* and *e'*.

Matchers are, in most cases, capable to determine the existence of a semantic relation between a pair of ontologies entities, and in some cases are even able to determine the existence of 1:n/n:1 semantic relation. However, no matcher is capable to determine the type of transformation occurring in the semantic relation.

4.4.2 Automatic Semantic Bridging

The result of similarity measure phase is a set of matches which denotes the confidence that a specific Matcher has about the similarity relation between source and target ontology's entities. The goal of this phase is transform the matches (level 0/1 alignment) provided by similarity measure phase into a valid ontology mapping document defined according to Semantic Bridge Ontology (SBO) (level 2 alignment). This process concerns with the definition of five distinct elements:

- Definition of Concept Bridges;
- Definition of Property Bridges;
- Definition of Relationships between Concept Bridges (sub bridge of relation);
- Definition of Relationships between Concept Bridges and Property Bridges (has bridge relation);
- Definition of Alternative Bridges and their relations with Concept Bridges and Property Bridges.

Defining semantic bridges is a very complex task comprehends the identification and specification of:

- The sets of source and target entities to bridge together;

- The transformation (function) holding between the source and the target entities sets (how data is translated);
- The conditions required for bridging the two set of entities (which conditions the data translation is performed).

The goal of this process is, using the result of similarity measure phase, combine the matches and transform it into a valid ontology mapping document that can be use to transform source ontology's instances in target ontology's instances. Beside that some heuristics is used to improve this process. Automatic ontology mapping process is a very complex task.

4.5 Conclusion

This chapter described the developed ontology mapping evolution process.

A set of ontology mapping evolution changes was defined (basic and composite changes) which are related between them expressed as a dependency matrix. The mapping evolution process was divided in two iterative sub processes: (i) correcting invalid SBO entities and (ii) processing the new ontology entities.

The main achievement of the work described in this chapter respects to the comprehension and systematization of the involved complex dimensions of the problem, such as:

- The SBO ontology;
- The MAFRA Toolkit Architecture and implementation;
- The ontology mapping process (implemented approach);
- Integrating new ontology entities processing and the correction of invalid SBO entities.

Notice that the processing of new ontology entities is not further addressed in this document, but it is an important open research issue. Future work on this subject will be pointed out in section 7.2

Next two chapters address the process of correcting the SBO entities, distinguishing the approaches depending on the availability of ontology evolution information.

Chapter 5

Correct Invalid SBO Entity

Using Ontology Evolution Information

This chapter is about the use of ontology evolution information to correct the invalid SBO entities. Knowing the meaning of an ontology change performed by the user is very useful for the automation of the ontology mapping evolution process.

Accordingly, this chapter aims to further address the subject introduced in previous section 4.3: the process of correcting invalid SBO entities using ontology evolution information provided by ontology evolution phase to correct the invalid SBO entities (Figure 24).

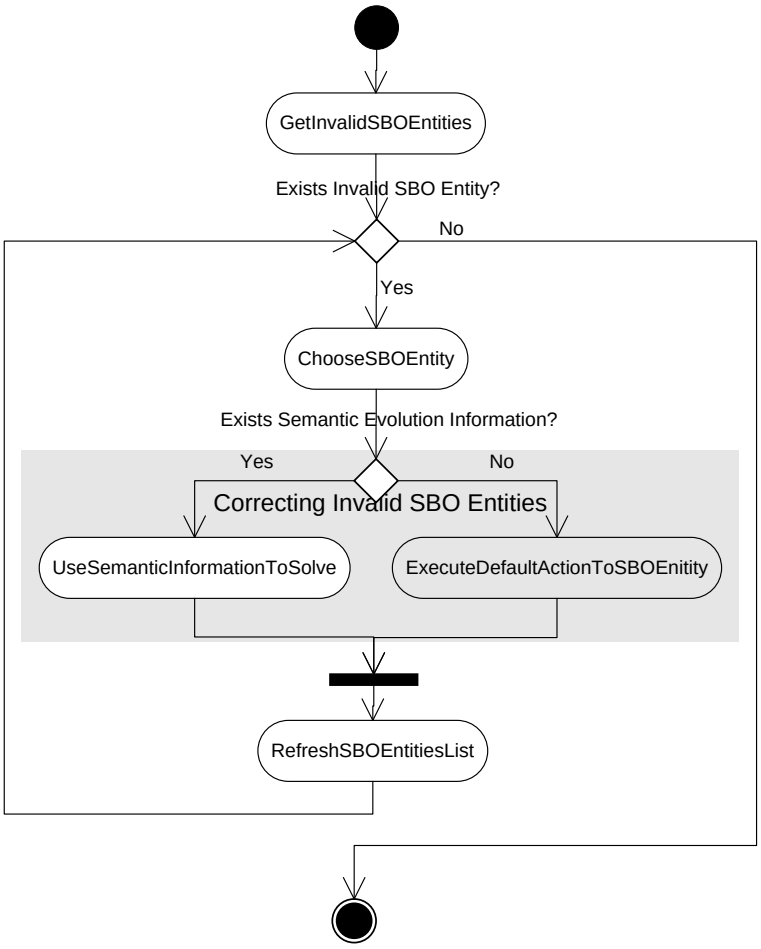

Figure 24 – Correcting invalid SBO entities (using ontology evolution information)

An ontology change requested by user is rarely executed as a single change. The application of an elementary change to ontology can induce inconsistencies in other parts of the ontology [Stojanovic, 2004]. One way for solving those inconsistencies is using ontology evolution strategies proposed by [Stojanovic, 2004], which is defined as "a method of finding the consistent ontology which meets the needs of the ontology engineer and by observing and analyzing the user's behavior in order to anticipate his needs". In [Stojanovic, 2004] the author presents a set of evolution strategies that can be selected by the user to meet his needs. For example: when a user wants to remove a concept from the ontology, one possible strategy defines that besides the concept, all its sub concepts are removed as well.

This chapter describes the process of exploiting information provided by ontology evolution phase during ontology mapping evolution process. Based on evolution strategies proposed by [Stojanovic, 2004], a set of ontology evolution scenarios was identified, which will support the automation of the ontology mapping evolution process. The idea is to exploit the concept of complex change for capturing user desires and behaviors, in benefice of the ontology mapping evolution process. These new complex changes are referred as ontology evolution scenarios.

Section 5.1 gives an introduction about ontology evolution strategies proposed by [Stojanovic, 2004], section 5.2 identifies a few ontology evolution scenarios according to ontology evolution strategies and in section 5.3 a bridge between ontology evolution scenarios and ontology mapping evolution scenarios is established.

5.1 Ontology Evolution Strategies

This section presents a set of evolution strategies as proposed by [Stojanovic, 2004] and how to identify them in an evolution log file provided by KAON tool[10]. Three concepts are especially relevant in this context:

- **resolution point** that represents a dilemma that might occurs during the changes resolution;
- **elementary evolution strategies** as a set of possible ways for resolving one resolution point;
- **evolution strategy** as a common policy consisting of a set of elementary evolution strategies, each giving an answer for one resolution point.

Some evolution strategy are used at instance level only (e.g. how to handle instances whose concept is deleted), which are not relevant for this work. Table 9 presents a set of the relevant ontology evolution strategies for this work.

Ontology Evolution Resolution Point	Elementary Ontology Evolution Strategies
Determines how to handle orphaned concepts	Orphaned concepts are deleted
	Orphaned concepts are reconnected to their parents
	Orphaned concepts are reconnected to the root concept
Determines how to propagate properties to the concept whose parents are changed	Don't propagate any properties of super concepts
	Propagate all properties (including the inherited properties) to the concept whose parent changes
	Propagate only properties of the parent concept to the concept whose parent changes

Table 9 – Part of resolution points and evolution strategy

The **how to handle orphaned concepts** resolution point occurs when the sub concept relationship between two concepts must be deleted. According to the dependency matrix of elementary change [Stojanovic, 2004], a relationship between two concepts should be deleted when one concept is deleted. The **how to propagate properties to the concept whose parents are changed** resolution point occurs when a parent of concept changes. In this case, one needs to know what to do to the properties of the concept.

[10] *http://kaon.semanticweb.org/*

5.1.1 How to Handle Orphaned Concepts

For this resolution point, three disjoint elementary evolution strategies are possible, illustrated in Figure 25, Figure 26 and Figure 27.

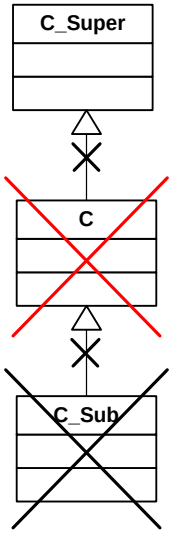

Figure 25 – Delete orphaned concept

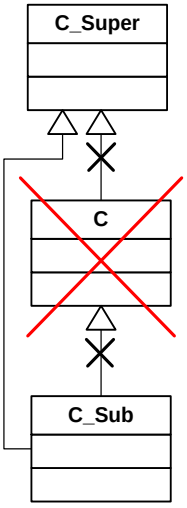

Figure 26 – Reconnect concept to super concept

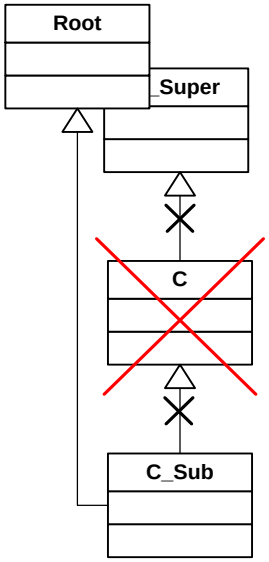

Figure 27 – Reconnect to root concept

In the examples presented before, the requested user change is to remove concept *C*. All others changes (removals) are consequences of the first one and dependent on the specific strategy.

5.1.1.1 Delete Orphaned Concept

Delete orphaned concept is the simplest solution and it is the only option for many ontology development tools. In this example, the following actions are taken as consequence of deleting concept *C*:

1. Delete concept *C_Sub*;
2. Remove concept *C_Sub* from sub concepts of *C*;
3. Remove concept C from sub concepts of *C_Super*.

5.1.1.2 Reconnect Concept to Super Concept

Reconnect concept to super concept is very useful when one wants to preserve hierarchy. In this example, the following actions are taken as consequence of deleting concept *C*:

1. Remove concept *C_Sub* from sub concepts of *C*;
2. Make concept *C_Sub* sub concept of *C_Super*;
3. Remove concept *C* from sub concept of *C_Super*.

5.1.1.3 Reconnect to Root Concept

Reconnect concept to root means leaving the concept *orphan*. In KAON ontology language, all concepts must always have a hierarchical path to the root concept. In this case, the following actions are taken as consequence of deleting concept *C*:

1. Remove concept *C_Sub* from sub concepts of *C*;
2. Make concept *C_Sub* sub concept of Root concept;
3. Remove concept *C* from sub concepts of *C_Super*.

5.1.2 How to Propagate Properties to the Concept whose Parent Changed

For this resolution point, three elementary evolution strategies are allowed. Consider the scenario in Figure 28 as example with the following entities and relations:

- **Concepts** – *C_Super*, *C* and *C_Sub*;
- **Properties** – *P_Super* (domain *C_Super*), *P* (domain *C*) and *P_Sub* (domain *C_Sub*);
- **Concept Hierarchical relation** – *C* sub class of *C_Super*, *C_Sub* sub class of *C*;
- **Property Hierarchical relation** – No hierarchical relation exists between P, *P_Sub* and *P_Super* properties.

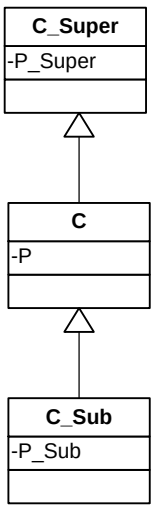

Figure 28 – Scenario example for properties propagation

Based on this scenario, consider now that the hierarchical relation between *C_Sub* and *C* is deleted. The problem to deal with in these evolution scenarios respect the consequences upon the properties of the concepts. The three possible elementary evolutions strategies are illustrated in Figure 29, Figure 30 and Figure 31, and further described in next sub-sections.

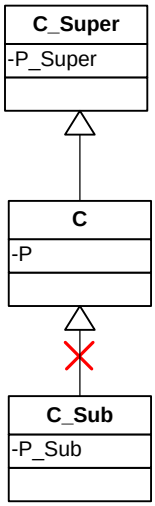

Figure 29 – Don't propagate any property

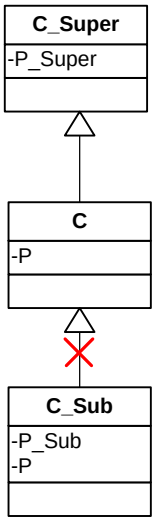

Figure 30 – Propagate properties from direct parent

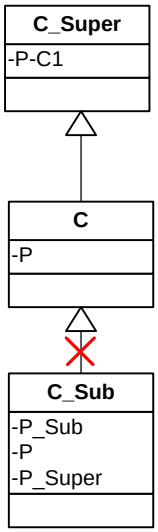

Figure 31 – Propagate all properties

5.1.2.1 Don't Propagate any Property

Don't propagate any property is the simplest solution, and the most used for many ontology development tools. No changes are fired as consequence of the first one Figure 29.

5.1.2.2 Propagate Properties from only Direct Super Concept

Propagate properties from direct parent means propagating only properties inherited from direct parents (not parents of parents). In this example, the property *P* will be propagated to concept *C_Sub* as consequence of removing the sub class of relation between *C_Sub* and *C* Figure 30.

5.1.2.3 Propagate all Properties

Propagate all properties means propagating all properties of parent concept, and all inherited properties from parent concept. In this example the *P* and *P_Super* properties will be propagated as consequence of removing the sub class of relation between *C_Sub* and *C* Figure 31.

5.2 Ontology Evolution Scenarios

After analyze the resolution points in previews section, one can notice that sometimes, more than one resolution point appears for one requested user change. Combining the example showed in 5.1.1 and 5.1.2, the decision for second resolution point depends on decision taken in first resolution point.

In the scenario depicted in Figure 32, when user deletes concept *C*, there is two resolution points (handle orphaned concepts and handle properties propagation) to choose an elementary evolution strategy.

Depending on decision in the first resolution point (how to handle orphaned concepts), the available elementary evolution strategies for the second is different.

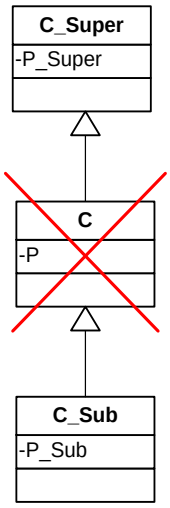

Figure 32 – Scenario example for relation between strategies

For the scenario presented in Figure 32, the consequent changes are those presented in the decision tree in Figure 33. Each node (except the root) represents an ontology evolution scenario, and each tree level represents a resolution point.

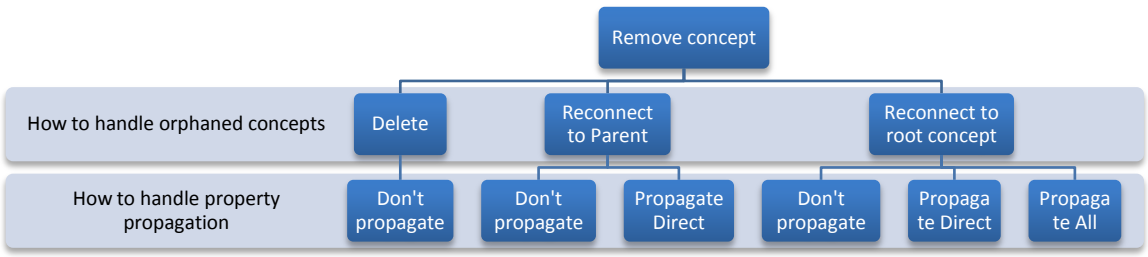

Figure 33 – Decision tree for combining ontology evolution strategies

As represented, the available elementary evolution strategies for the second resolution point (how to propagate properties to the concept whose parents are changed) depends on the elementary evolution strategy selected in the first resolution point (how to handle orphaned concepts).

In KAON ontology evolution approach (see section 3.2) there is an evolution log that records the exact sequence of changes that occurred when an ontology engineer updated the ontology. This evolution log contains a set of elementary changes related between them as change and consequent relation.

The identification and characterization of each ontology evolution scenario was made using the following steps:

1. Accessing the evolution log file to read the changes;

2. Identify the necessary and sufficient conditions;
3. Identify the necessary and sufficient contextual conditions.

The following sub sections describe these steps.

5.2.1 Accessing the Evolution Log

Many ontology editors use a log file to record information about ontology changes. In the scope of this work, accessing the evolution log file is made using the KAON API [Maedche, et al., 2003], namely because:

- KAON provides a specific API for reading, writing, and modifying the evolution log file;
- MAFRA Toolkit uses KAON API as core ontology management technology.

The KAON API is a set of interfaces developed by the Karlsruhe University[11] in order to offer programmatic access to KAON ontologies, and for accessing evolution log, by providing classes such as Concept, Property and Instance [Gabel, et al., 2004].

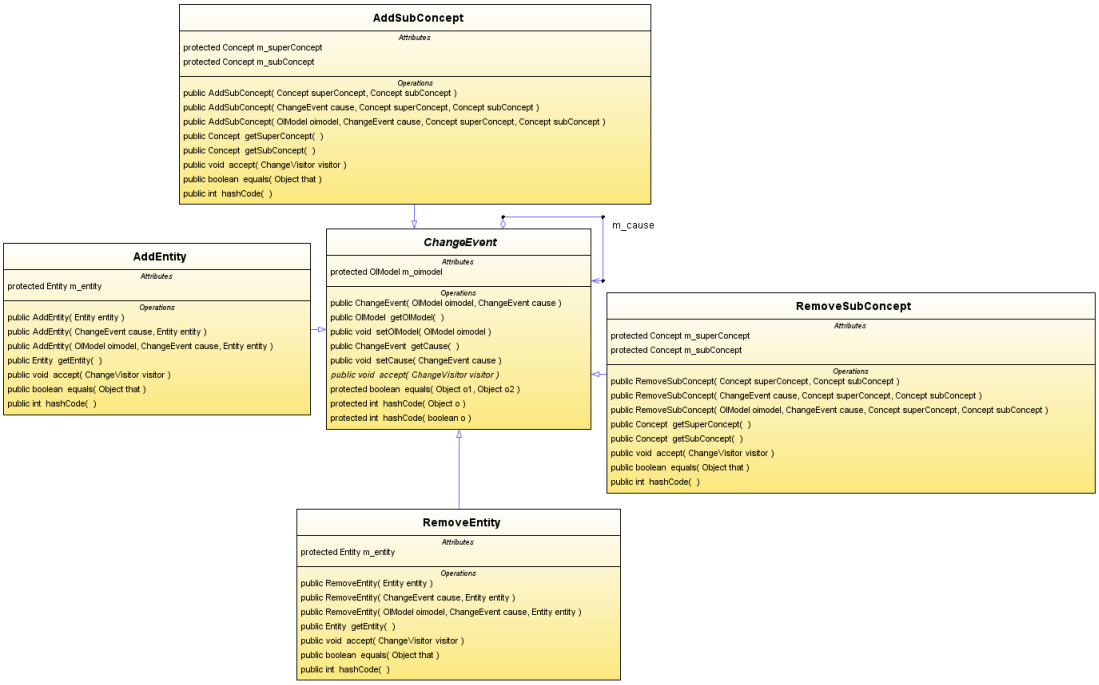

Figure 34 – KAON API – Part of Evolution Class Diagram

Figure 34 represents some of the classes used to manage ontology changes. As depicted:

- The super class *ChangeEvent* represents a generic ontology change;

[11] *http://www.uni-karlsruhe.de/*

- ChangeEvent's sub classes represent specifics ontology changes as described in Table 1 in section 3.2.2.1 (Elementary Changes in KAON ontology evolution approach);
- Causes and consequences are modeled through the cause relation established at the ChangeEvent class;
- Specific classes of changes require specifics arguments (e.g. RemoveSubConcept has super and sub concept as argument).

The evolution log file can be seen as a graph of change events where one ontology change event is related to others as its consequences.

Using the KAON API, one can retrieve all the changes performed to the original ontology. As consequence, it is possible to iteratively generate (forward and backward) snapshots (versions) of the ontology through the entire evolution process.

5.2.2 Identify the Necessary and Sufficient Conditions

In order to identify a specific ontology evolution scenario, it is necessary to identify the set of necessary and sufficient conditions. In this section, according to the resolution point and respective elementary evolution strategy presented in the decision tree depicted in Figure 33, a set of the necessary and sufficient conditions for each ontology evolution scenario is presented.

In the following figures, each node represents the change events that are triggered as consequence of the execution of its parent change. Between brackets are the change events not mandatory.

5.2.2.1 Delete Orphaned Concepts Facts

The set of sufficient and necessary conditions for identifying a "Delete Orphaned Concept" scenario is presented in Figure 35.

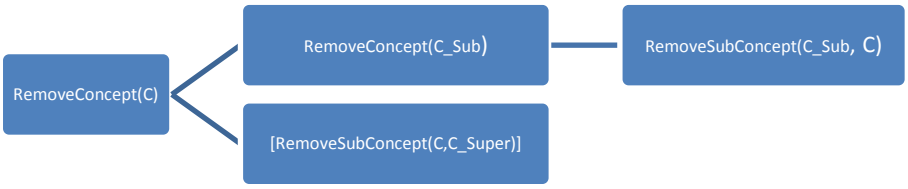

Figure 35 – Delete orphaned concept Facts

Starting from the *RemoveConcept(C)* change user request, if both:

- RemoveConcept(*C_Sub*) change exists as consequence of RemoveConcept(C);
- RemoveSubConcept(*C_Sub*, C) change exists as consequence of RemoveConcept(*C_Sub*),

then it is possible to conclude that the followed ontology evolution strategy was “Delete Orphaned Concept”. Notice therefore, that the *RemoveSubConcept(C,C_Super)* change is not mandatory for identifying this scenario.

5.2.2.2 Reconnect Orphaned Concept to Root Facts

The set of sufficient and necessary conditions for identifying a “Reconnect Orphaned Concept to Root” scenario is presented in Figure 36.

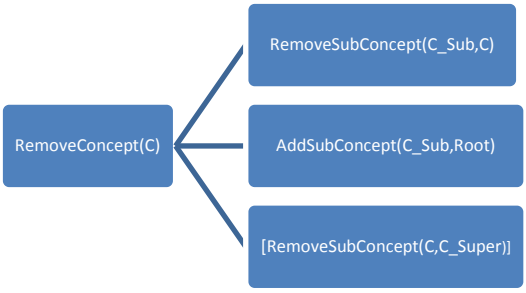

Figure 36 – Reconnect Orphaned Concept to Root Facts

Starting from the user requested change *RemoveConcept(C)*, if both:

- *RemoveSubConcept(C_Sub,C)* change exists as consequence of *RemoveConcept(C)*;
- *AddSubConcept(C_Sub, Root)* change exists as consequence of *RemoveConcept(C)*,

then it is possible to conclude that the followed evolution strategy was “Reconnect Orphaned Concept to Root”. Again, notice that the *RemoveSubConcept(C,C_Super)* change is not mandatory for indentifying this scenario.

5.2.2.3 Reconnect Orphaned Concept to Parent Facts

The set of sufficient and necessary conditions for identifying a reconnect orphaned concept to parent scenario is presented in Figure 37.

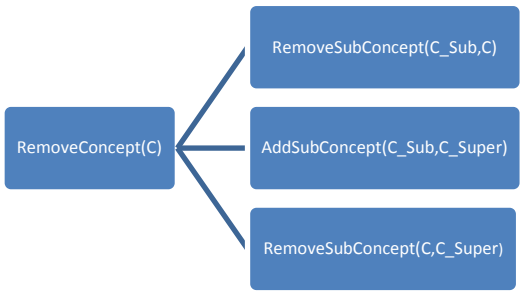

Figure 37 – Reconnect Orphaned Concept to Parent Facts

Starting with *RemoveConcept(C)* action requested by the user, if all:

- *RemoveSubConcept (C_Sub,C)*;
- *AddSubConcept(C,C_Super)*;
- *RemoveSubConcept(C,C_Super)*,

changes exist as consequence of *RemoveConcept(C)*, then it is possible to conclude that the followed ontology evolution scenario is "Reconnect Orphaned Concept to Parent".

5.2.2.4 Don't Propagate any Property Domain Facts

The set of necessary and sufficient conditions for identifying a "Don't propagate any property domain" scenario is presented in Figure 38. The "!"symbol before change event means absent of the change event.

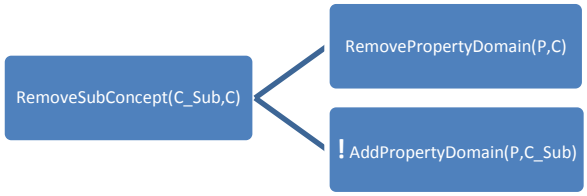

Figure 38 – Don't Propagate any Property Domain Facts

Starting from *RemoveSubConcept(C_Sub,C)* change requested by the user, if both:

- *RemovePropertyDomain(P,C)* change exists as consequence of *RemoveSubConcept(C_Sub,C)*;
- *AddPropertyDomain(P,C_Sub)* change does not exist as consequence of *RemoveSubConcept(C_Sub,C)*,

then it is possible to conclude that the followed ontology evolution scenario is "Don't Propagate any Property Domain".

5.2.2.5 Propagate Properties Facts

The set of necessary and sufficient conditions for identifying a "Propagate all properties or Propagate only direct property" scenario is presented in Figure 39.

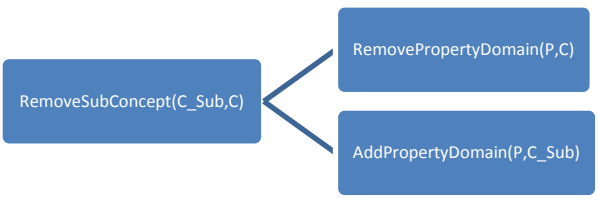

Figure 39 – Propagate properties Facts

The evolution changes presented in Figure 39 allows identifying two distinct ontology evolution scenarios:

- All properties are propagated (including inherited from super concepts);
- Propagates direct properties only.

Distinctions between these two ontology evolution scenarios depend on the difference between the number of *RemovePropertyDomain* (#*RemovePropertyDomain*) and the number of *AddPropertyDomain (#AddPropertyDomain)* events.

Systematizing:

- #*RemovePropertyDomain* < #*AddPropertyDomain* → All properties are propagated;
- #*RemovePropertyDomain* = #*AddPropertyDomain* → Propagates direct properties only.

5.2.3 Identify the Necessary and Sufficient Contextual Conditions

Besides the evolution change conditions, it is necessary to identify and check the necessary contextual conditions. There are two types of contextual conditions:

- **Ontology Conditions** – Necessary conditions upon ontologies. Even if all conditions characterizing an ontology evolution scenario are true, it is necessary to check the correct existence of all involved entities in that scenario (e.g. need to check if all entities still exist as they can be removed by others actions);
- **Ontology Mapping Conditions** – Necessary conditions in ontology mapping document. To apply an ontology mapping scenario some conditions in ontology mapping document need to be verified (e.g. existence of some SBO entities and some relations between them or changes don't violate the SBO language restrictions). These conditions depend on the scenario.

5.3 Ontology Evolution Scenarios for Mapping Evolution Scenarios

This section describes the proposed approach for exploiting information from ontology evolution phase, to improve the ontology mapping evolution process using ontology evolution scenarios. Accordingly, a set of corresponding ontology mapping scenarios is presented.

In the following sections a specific UML-based notation will be used. This notation is depicted in Figure 40, using some examples.

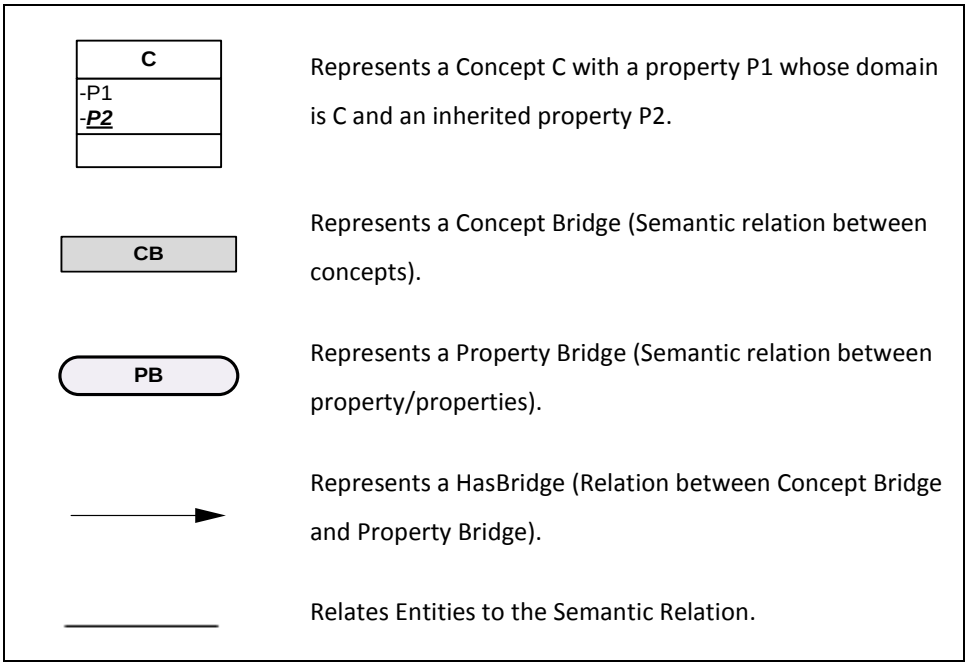

Figure 40 – Ontology mapping notation

Using the notation in Figure 40, Figure 41 illustrates a valid ontology mapping scenario, relating entities from O1 and O2 ontologies before any ontology evolution.

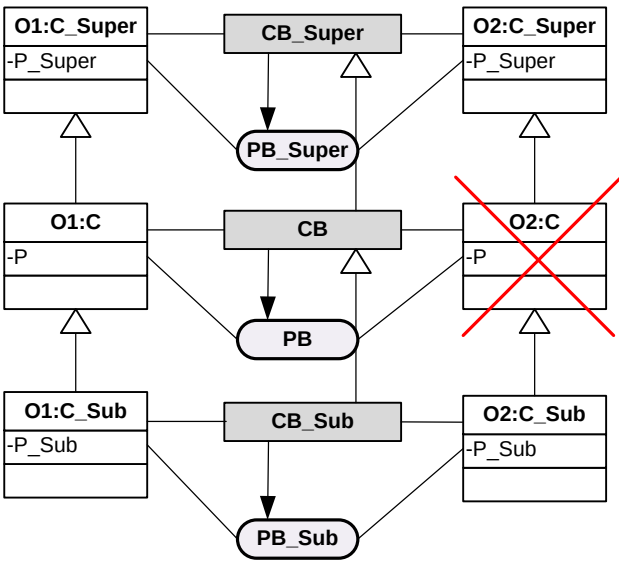

Figure 41 – Ontology evolution scenario for mapping evolution scenario

***O1* ontology entities:**

- **Concepts** – *O1:C_Super*, *O1:*C (sub class of *O1:C_Super*) and *O1:C_Sub* (sub class of *O1:*C);
- **Properties** – *O1:P_Super* (domain *O1:C_Super*), *O1:*P (domain *O1:C*) and *O1:P_Sub* (domain *O1:C_Sub*).

***O2* ontology entities:**

- **Concepts** – *O2:C_Super*, *O2:C* (sub class of *O2:C_Super*) and *O2:C_Sub* (sub class of *O2*:C);
- **Properties** – *O2:P_Super* (domain *O2:C_Super*), *O2:*P (domain *O2*:C) and *O2:P_Sub* (domain *O2:C_Sub*).

Mapping document:

- **Concept Bridges** – *CB_Super* (relating *O1:C_Super* and *O2:C_Super*), *CB* (relating *O1:*C and *O2:*C) and *CB_Sub* (relating *O1:C_Sub* and *O2:C_Sub*);
- **Property Bridges** – PB_Super (relating *O1:P_Super* and *O2:P_Super*, in context of *CB_Super*), PB (relating *O1:*P and *O2:*P, in context of *CB*) and PB_Sub (relating *O1:P_Sub* and *O2:P_Sub*, in context of *CB_Sub*).

Assuming that ontology engineer requested a *RemoveConcept(C)* change, there will be two resolution points to solve: (i) how to handle orphaned concept and (ii) how to propagate properties to the concept whose parents are changed. According to the elementary evolution strategy chosen by the user, additional changes are performed to keep ontology consistent. The resolution of the second resolution point depends on the first.

Based on the decision tree of the ontology evolution scenarios (Figure 33) presented in section 5.2, correspondences to the ontology mapping scenarios are established. The following sections present the combination of elementary evolutions strategy for the two resolution points, according to the decision tree of ontology evolution scenarios. Figure 42 identifies the 5 different strategies, arising from the decision tree first presented in Figure 33.

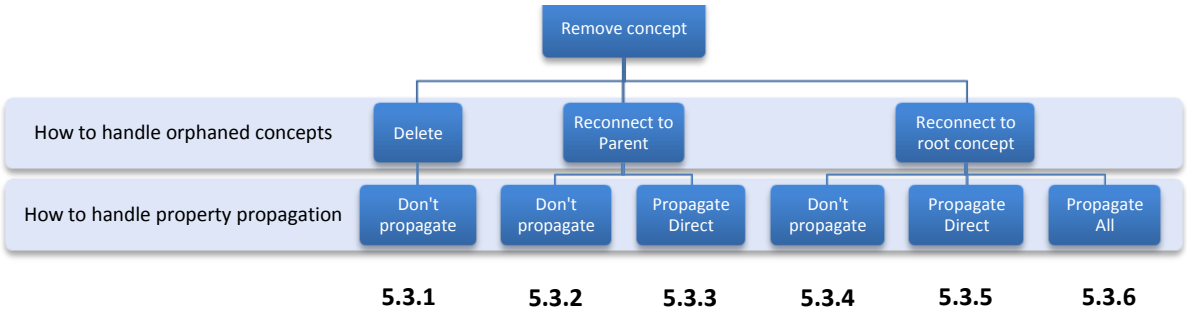

Figure 42 – Ontology evolution scenarios

5.3.1 Delete Orphaned Concept and Don't Propagate any Properties

This strategy is applied in case the user wants to remove the orphaned concept (*O2:C_Sub*) and consequently no properties are propagated. As consequence, the following entities become invalid (see section 4.3.1):

- The Concept Bridge *CB* becomes invalid because its target concept value no longer exists in *O2* ontology;

- The Concept Bridge *CB_Sub* becomes invalid because its target concept value no longer exists in *O2* ontology;
- The Property Bridge PB becomes invalid since its target argument *O2:*P no longer exist;
- The Property Bridge PB_Sub becomes invalid since its target argument *O2:P_Sub* no longer exist.

To solve these inconsistencies, and considering that delete orphaned concept strategy was used, the mapping scenario depicted in Figure 43.

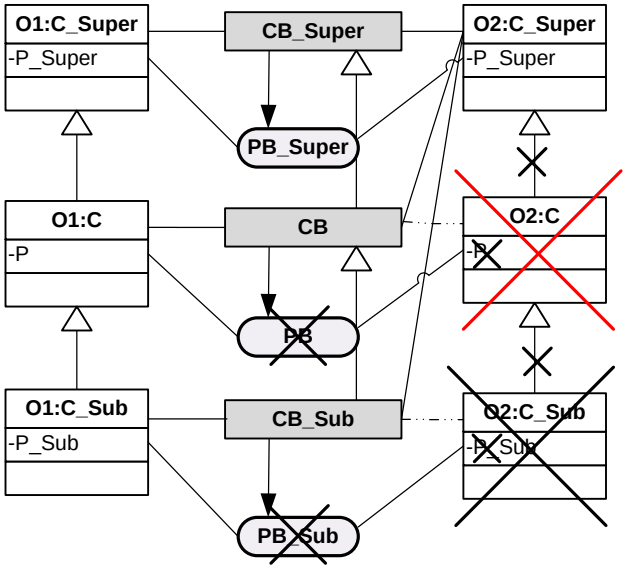

Figure 43 – Mapping scenario when delete orphaned concept and don't propagate any properties

In this case, the following mapping changes will be taken (Figure 43):

- Concept Bridge *CB* changes its target concept value to *O2:C_Super*. This action is taken because all instances of *O2:*C are also instances of *O2:C_Super*;
-
- Concept Bridge *CB_Sub* changes its target concept value to *O2:C_Super*. This action is taken because all instances of *O2:C_Sub* is also instance of *O2:C_Super*;
- All Property Bridges of *CB* are deleted since the properties no longer exist;
- All Property Bridges of *CB_Sub* are deleted since the properties no longer exist.

5.3.2 Reconnect Concept to Super Concept and Don't Propagate any Property

This strategy is applied when the orphaned concept (*O2:C_Sub*) is reconnected to the super concept (*O2:C_Super*) and no properties were propagated. As consequence, the following SBO entities become invalid:

- Concept Bridge *CB* will be invalid, because its target concept argument no longer exists in *O2* ontology;

- The Property Bridge PB its target argument value *O2:*P no longer exists in ontology.

To solve these inconsistencies, and considering the applied strategy, the mapping scenario depicted in Figure 44 is proposed.

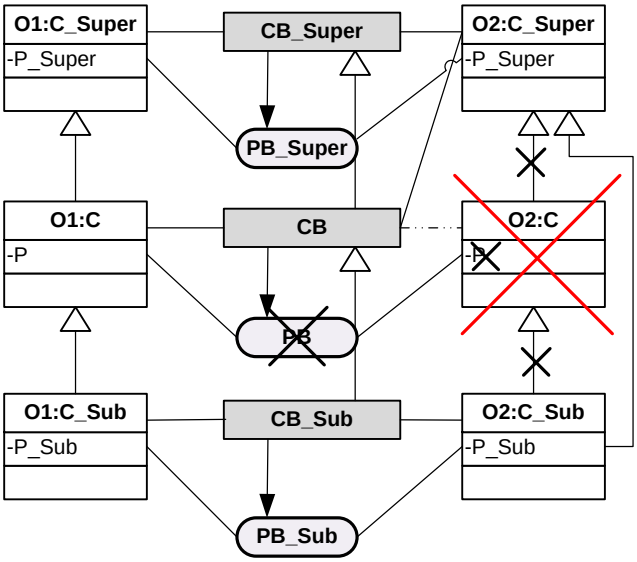

Figure 44 – Mapping scenario: reconnect to super concept and don't propagate any properties

In this case, the following mapping changes will be taken (Figure 44):

- Concept Bridge *CB* changes its target concept argument to *O2:C_Super*, based on the fact that all instances of *O2:*C are also instances of *O2:C_Super*;
- All Property Bridges of *CB* are deleted since the properties were deleted.

5.3.3 Reconnect Concept to Super Concept and Propagate only Direct Properties

This strategy is applied when the orphaned concept (*O2:C_Sub*) is reconnected to the super concept (*O2:C_Super*) and only direct properties are propagated. As consequence the following SBO entities become invalid:

- Concept Bridge *CB* will be invalid because the target concept argument no longer exists in *O2* ontology;
- Property Bridge PB becomes invalid since its target argument value no longer exists.

To solve these inconsistencies, and considering the applied strategy, the mapping scenario depicted in Figure 45 is proposed.

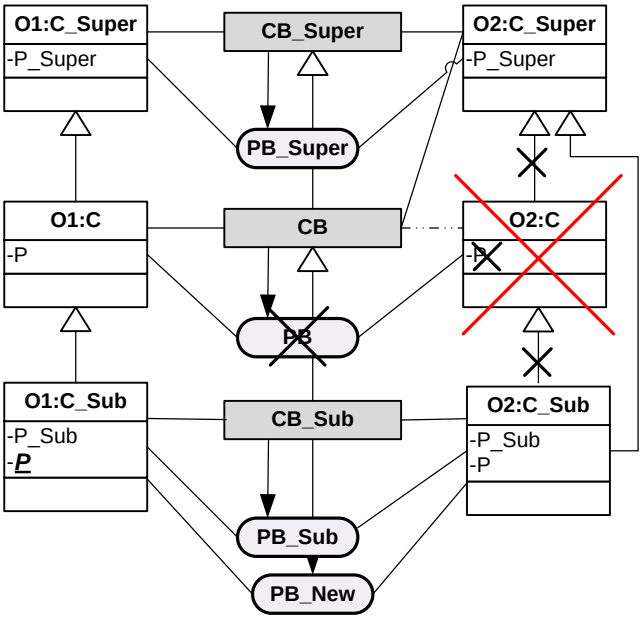

Figure 45 – Mapping scenario: reconnect to super concept and propagate direct properties

In this case, the following mapping changes will be taken (Figure 45):

- Concept Bridge *CB* changes its target concept value to *O2:C_Super*, based on the fact that all instances of *O2:*C are also instances of *O2:C_Super*;
- Property Bridges of *CB* are propagated to *CB_Sub* and deleted from *CB* context) since the properties of C were propagated to *C_Sub*.

5.3.4 Reconnect Orphaned Concept to Root and Don't Propagate any Property

This strategy is applied when the orphaned concept (*O2:C_Sub*) is reconnected to the Root concept and no property was propagated. As consequence the following SBO entities becomes invalid:

- Concept Bridge *CB* will be invalid because the target concept value no longer exists in *O2* ontology;
- Property Bridge PB becomes invalid since its target argument value no longer exists.

To solve these inconsistencies, and considering the applied strategy, the mapping scenario depicted in Figure 46 is proposed.

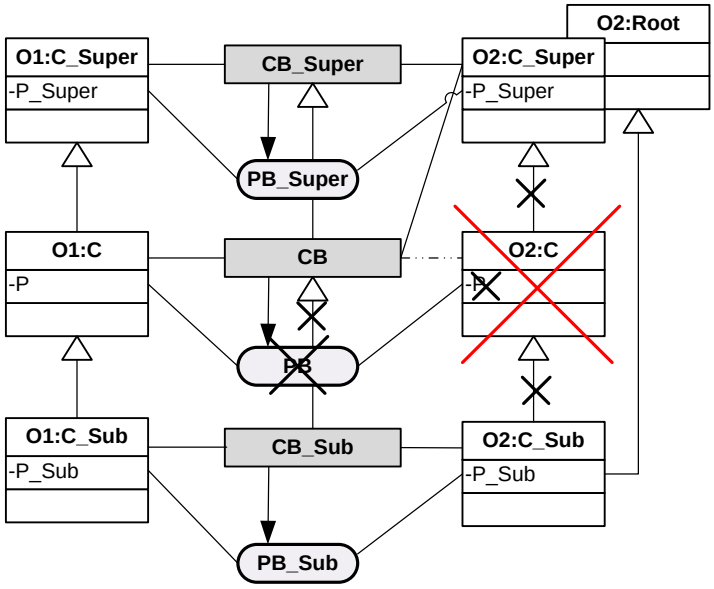

Figure 46 – Mapping scenario: reconnect to root and don't propagate property

For this scenario, the following actions in taken:

- Concept Bridge *CB* changes its target concept value to *O2:C_Super* based on the fact that all instances of *O2:*C are also instances of *O2:C_Super*;
- All Property Bridges of *CB* are deleted since the properties no longer exist;
- The subBridgeOf relation between *CB* and *CB_Sub* is deleted because there is no hierarchical relation between *C_Super* and *C_Sub*.

5.3.5 Reconnect Orphaned Concept to Root and Propagate direct Properties

This strategy is applied when the orphaned concept (*O2:C_Sub*) is reconnected to the Root concept and only direct property was propagated. As consequence the following SBO entities becomes invalid:

- Concept Bridge *CB* will be invalid because the target concept value no longer exists in *O2* ontology;
- Property Bridge PB becomes invalid since its target argument value no longer exists.

To solve these inconsistencies, and considering the applied strategy, the mapping scenario depicted in Figure 47 is proposed.

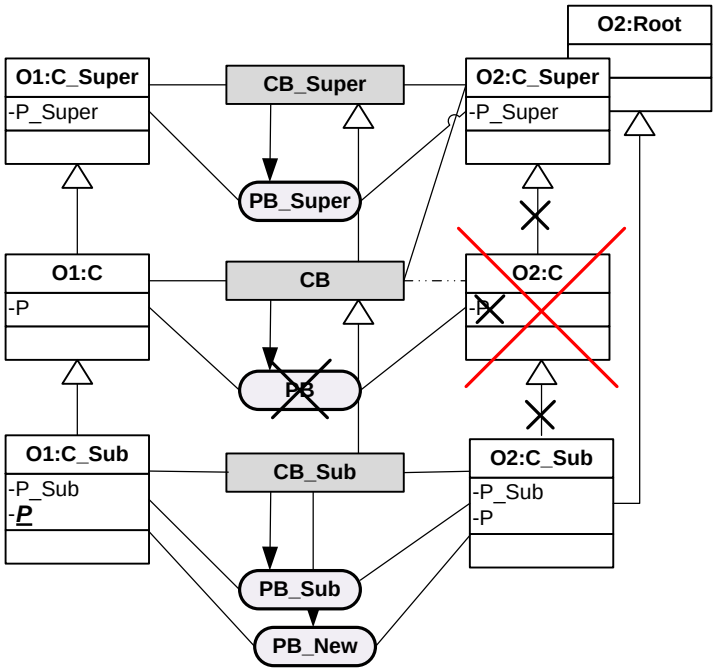

Figure 47 – Mapping scenario: reconnect to root and propagate only direct properties

In this case, the following mapping changes will be taken (Figure 47):

- Concept Bridge *CB* changes its target concept value to *O2:C_Super* based on the fact that all instances of *O2:*C are also instances of *O2:C_Super*;
- Property Bridges of *CB* are propagated to *CB_Sub* and deleted from *CB CB_Sub* context since the properties of C wer propagated to *C_Sub*;
- The subBridgeOf relation between *CB* and *CB_Sub* is deleted because there is no hierarchical relation between *C_Super* and *C_Sub*.

5.3.6 Reconnect Orphaned Concept to Root and Propagate all Properties

This strategy is applied when the orphaned concept (*O2:C_Sub*) is reconnected to the Root concept and all properties was propagated. As consequence the following SBO entities become invalid:

- Concept Bridge *CB* will be invalid because the target concept value no longer exists in *O2* ontology;
- Property Bridge PB becomes invalid since its target argument value no longer exists.

To solve these inconsistencies, and considering the applied strategy, the mapping scenario depicted in Figure 48 is proposed.

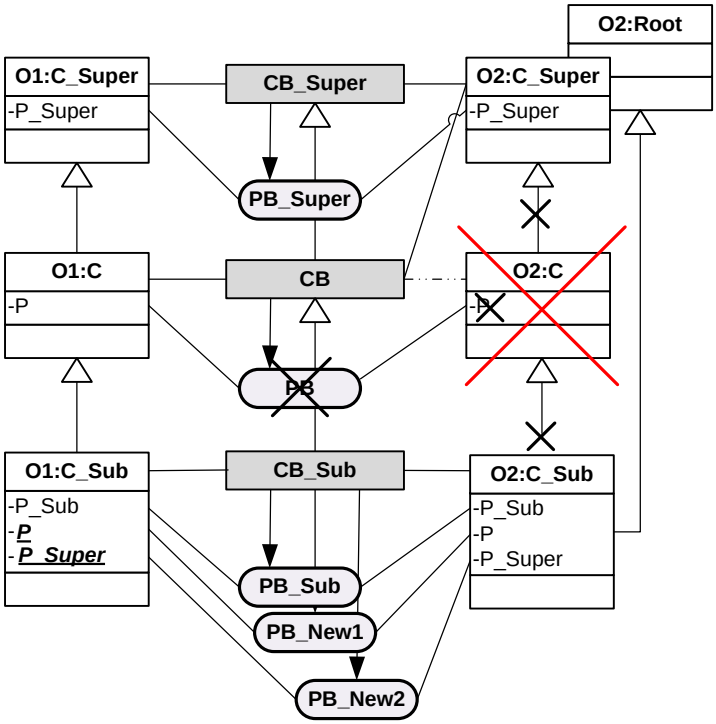

Figure 48 – Mapping scenario: reconnect to root and propagate all properties.

In this case, the following mapping changes will be taken (Figure 48):

- Concept Bridge *CB* changes its target concept value to *O2:C_Super* based on the fact that all instances of *O2:*C are also instances of *O2:C_Super*;
- Property Bridges of *CB* are propagated to *CB_Sub* and deleted from *CB* context since the properties of C were propagated to *C_Sub*;
- Property Bridges inherited for *CB* are propagated to *CB_Sub* since the inherited properties of C were propagated to *C_Sub*;
- The sub bridge of relation between *CB* and *CB_Sub* is deleted because there is no hierarchical relation between *C_Super* and *C_Sub*.

5.4 Conclusion

This chapter proposed an approach for using semantic information provided by the ontology evolution process. The idea was to identify a set of ontology evolution scenarios and to use this information to improve the ontology mapping evolution process and in particular the phase of correcting invalid SBO entities.

First, the set of evolution strategies proposed by [Stojanovic, 2004] were analyzed and exploited to capture ontology evolution scenarios. After the scenarios identification, using a set of necessary and sufficient conditions to interpret them in an evolution log file, correspondences to ontology mapping scenarios were established.

Technologically, this section addressed the following issues:

- Analysis of the evolution log provided by KAON tool;
- Analysis of the KAON API and the KAON evolution ontology to access the evolution log;
- Identification of the evolution scenarios through the evolution log.

However, notice that the work described in this chapter is constrained in several dimensions:

- The process is limited to the identified scenarios;
- The system uses the log provided by KAON tool only;
- The used ontology language is RDFS, and the description logic component of OWL is not exploited. This is primarily due to limitations of MAFRA Toolkit and SBO language, as both are unable to use other ontology language than RDFS.

Accordingly, complementary topics of research and development are identified and suggest new directions.

Finally, this chapter was useful to describe the potential of the information provided by the ontology development systems and evolution methodologies in particular. This information should be exploited to achieve new mapping document according to the user's preferences during the ontology evolution phase.

Chapter 6

Correct Invalid SBO Entity

Without Ontology Evolution Information

Often ontology evolution information is not available during the ontology mapping evolution process, preventing an explicit alignment between the user's ontology evolution intentions and the ontology mapping evolution process.

This chapter presents an approach to deal with invalid mapping documents when ontology evolution information is unavailable. This process is the alternative for correcting invalid SBO entities using evolution information (Figure 49).

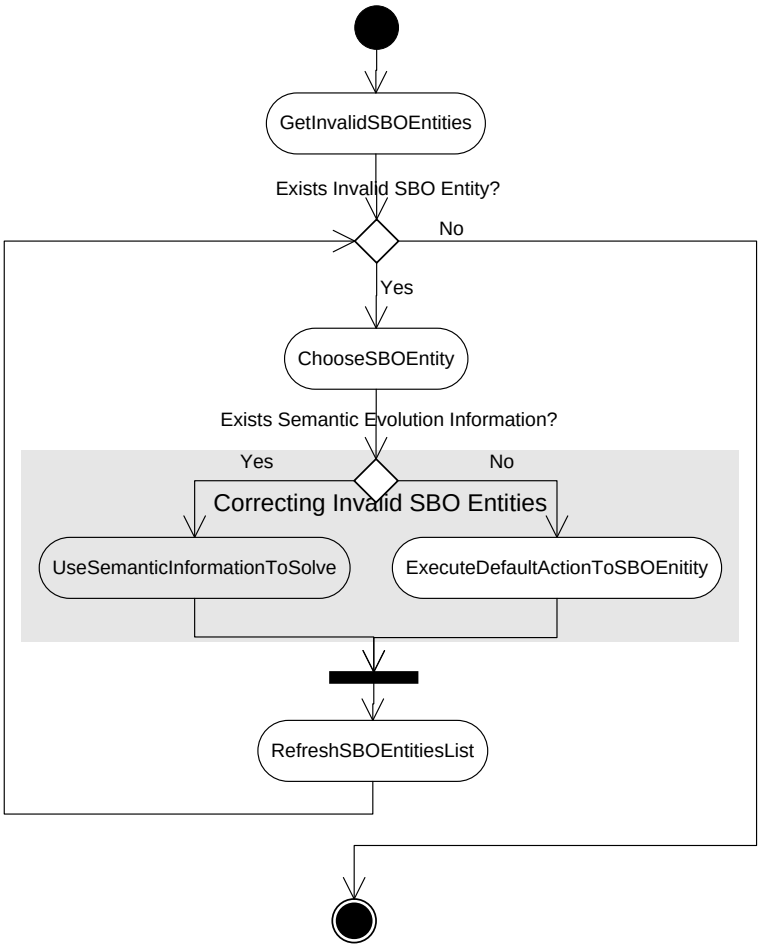

Figure 49 – Correcting invalid SBO entities (no ontology evolution information)

A semi-automatic ontology mapping evolution process based on the user parameterization of the system is proposed in this chapter. For that, a set of possible ways to correct each invalid situation were analyzed and systematized according to the reasons that turn it invalid. Complementarily, a set of ontology mapping evolution strategies is proposed.

6.1 Analysis

There are two types of invalid situations during ontology evolution process:

- Those that result from an ontology change;
- Those that result from the corrections of the previous changes.

Next two sections analyze these two situation types.

6.1.1 Ambiguities During Mapping Correction

Ambiguities occur during mapping correction when for an invalid mapping situation, several corrective mapping changes are possible, but none is explicitly the "best". Taking into account the invalid situations identified in section 4.3.1, Table 10 and Table 11 present for each invalid situation, the corrective evolution changes.

Invalid Concept Bridge situation	Corrective mapping change
Source AND Target Concept Value are Invalid	- Remove Concept Bridge[12]
Source XOR Target Concept Value is Invalid	- Remove Concept Bridge - Change Concept Bridge Concept Value (to super concept)
Extensional Specification is Invalid	- Remove Concept Bridge - Remove Invalid Condition
Generic Condition is Invalid	- Remove Concept Bridge - Remove Invalid Condition

Table 10 – Available mapping changes for invalid Concept Bridge

Invalid Property Bridge situation	Corrective mapping change
Source AND Target Argument Values are Invalid	- Remove Property Bridge[13]
Source XOR Target Argument Values are Invalid	- Remove Property Bridge - Change Property Bridge Argument Value
Extensional Specification is Invalid	- Remove Property Bridge - Remove Invalid Condition
Generic Condition is Invalid	- Remove Concept Bridge - Remove Invalid Condition

Table 11 – Available mapping changes for invalid Property Bridge

Accordingly, except the first situation of each of previous tables, all other situations are ambiguous.

A corrective mapping change is a single (elementary or composite) mapping change. Since an ontology mapping is a set of semantic bridges related between them (e.g. Concept Bridges have sub Concept Bridges and Property Bridges), an execution of a single ontology mapping change (e.g. remove a Concept Bridge) may cause inconsistencies in others SBO entities (e.g. subBridgeOf relation) and some other changes need to be performed to solve such problems. However, during this process, several paths for solving inconsistencies may be found.

6.1.2 Ambiguities During Mapping Change Resolution

A mapping change resolution is a process whereby after the execution of a single mapping change, all others necessary mapping changes are performed for solving inconsistencies in mapping document. Again, during this process, ambiguities for solving inconsistencies may be found.

---

[12] *This situation in not ambiguous since one single mapping corrective change is possible.*

[13] *This situation in not ambiguous since one single mapping corrective change is possible.*

Consider the corrective mapping changes presented in previous section:

- Remove Concept Bridge;
- Change Concept Bridge Concept Value;
- Remove Property Bridge;
- Change Property Bridge Argument Value;
- Remove Invalid Condition.

The following corrective changes do not cause inconsistencies in others semantic bridges:

- Removing a Property Bridge, fires the removal of the HasBridge relation to the attached Concept Bridges, but no SBO entities becomes invalid;
- Changing the Property Bridge Argument Value, does not fire any additional mapping change during the change resolution process;
- Remove Invalid Condition does not fire any additional mapping change during the change resolution process.

Therefore, none of these changes will be considered here after. Instead, the other mapping changes may cause some inconsistencies:

- Removing a Concept Bridge, fires the removal of HasBridge relation to the attached Property Bridges making them orphan;
- Removing a Concept Bridge also, fires the removal of subBridgeOf relation to the attached sub Concept Bridges making them orphaned;
- Changing Concept Bridge Concept Value may cause inconsistencies in the Property Bridge's argument path, because the bridged properties may not have the new concept value as domain.

Based on this, Table 12 summarizes the corrective mapping changes and the SBO entities that are affected.

Corrective Mapping Change	SBO entities that may become inconsistent
Remove Concept Bridge	- Property Bridges, after hasBridge relation - Sub Concept Bridges, after subBridgeOf relation
Change Concept Bridge Concept Value	- Property Bridges, after the arguments' paths

Table 12 – Corrective mapping change and inconsistencies

Next sections analyze the inconsistencies raised by these corrective mapping changes.

6.1.2.1 Ambiguities During Remove Concept Bridge

As consequence of the "Remove Concept Bridge" corrective mapping change, two potentially invalid and ambiguous situations might occur. Consider the mapping scenario in Figure 50.

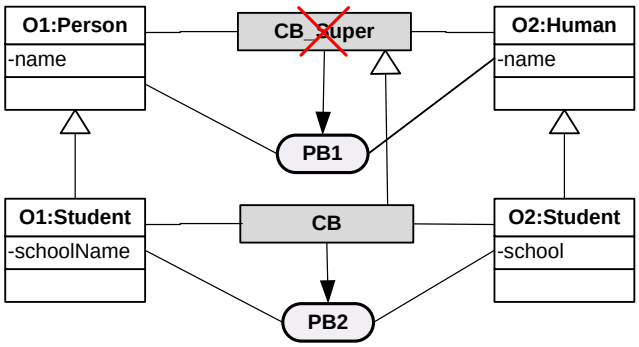

Figure 50 – Example ambiguities during Remove Concept Bridge

Supposing that the *CB_Super* Concept Bridge is removed, two potentially ambiguous situations arise:

- Property Bridge (PB1) becomes orphan, because the Concept Bridge (*CB_Super*) it was attached to via the hasBridge relation is removed;
- Concept Bridge (*CB*) becomes orphan, because the Concept Bridge (*CB_Super*) it was attached to via the subBridgeOf relation is removed.

Each of these situations has several resolution mapping changes possibilities (Table 13):

Ambiguous mapping situation	Resolution mapping change
Orphaned Property Bridge	- Don't Propagate any Property Bridges (Remove Property Bridges) - Propagate direct Property Bridges to sub Concept Bridges - Propagate (All) Property Bridges to sub Concept Bridges
Orphaned Concept Bridge	- Delete orphaned Concept Bridge - Reconnect orphaned Concept Bridge to parent - Keep orphaned Concept Bridge

Table 13 – Ambiguous mapping change during removing Concept Bridge

6.1.2.2 Ambiguities During Change Concept Bridge Concept Value

As consequence of the "Change Concept Bridge Concept Value" corrective mapping change, two potentially invalid and ambiguous situations might occur (Figure 50).

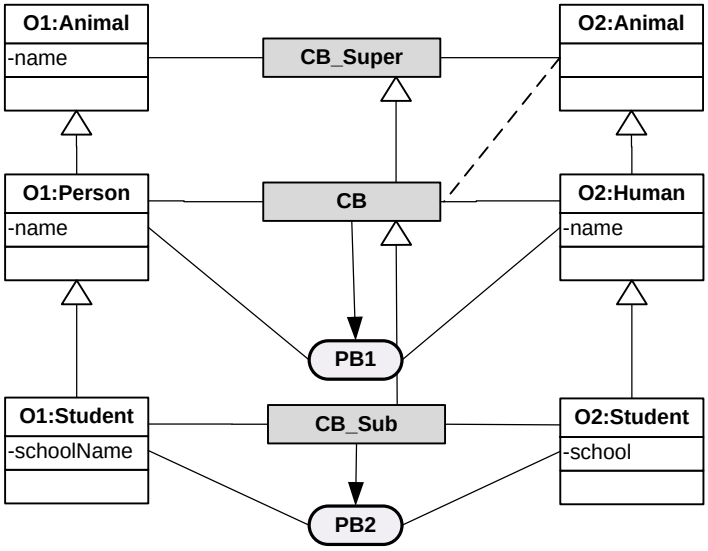

Figure 51 – Example ambiguities during Change Concept Bridge Concept Value

Changing the target concept value of *CB* Concept Bridge from o *O2:Human* to *O2:Animal* makes *PB1* become inconsistent since the property *O2:*name does not have *O2:*Animal as domain. This is an ambiguous situation since there are two solutions to deal with the inconsistent Property Bridge (Table 14).

Ambiguous Mapping Situation	Resolution Mapping Change
Orphaned Property Bridge	- Don't Propagate any Property Bridges (Remove Property Bridges) - Propagate direct Property Bridges to sub Concept Bridges

Table 14 – Ambiguous mapping situation and resolution point

As consequence, in every observed ambiguous situation, one and only one resolution change must be adopted. Unfortunately, no information is available to help the system deciding.

These ambiguities situations promote the concept of **resolution point** and **evolution strategies** similar to those proposed in scope of ontology evolution [Stojanovic, 2004].

6.1.3 Ontology Mapping Evolution strategies

As referred, during the ontology mapping evolution process, different ways to resolve the same mapping invalid situation are possible. However, none is either explicitly correct or wrong, since no ontology evolution information is available for disambiguation. These ambiguities promote the concept of mapping evolution strategy, similar to those used in scope of ontology evolution process.

In scope of ontology evolution, [Stojanovic, 2004] presents the following concepts:

- **Ontology resolution point** – Represents one dilemma that might occur during the resolution change in ontology evolution process;

- **Ontology elementary evolution strategies** – Set of possible ways for resolving one resolution point;
- **Ontology evolution strategy** – Common policy consisting of a set of elementary evolution strategies, each giving an answer for one resolution point.

Analogous to these concepts the following concepts are introduced:

- **Mapping evolution resolution point** (**MERP**) – representing a dilemma that might occur during the changes resolution in ontology mapping evolution process;
- **Mapping evolution elementary strategies** – representing a set of possible ways to resolve one mapping resolution point;
- **Mapping evolution strategy** – Pair between one mapping resolution point and one mapping evolution elementary strategy.

Systematizing the previous analysis (see previous sections), the invalid situations can be reduced to five different mapping scenarios which have the same available mapping evolution corrective actions (Table 15).

ID	Mapping Evolution Resolution Point	Mapping Evolution Elementary Strategy
MERP1	How to handle invalid Source OR Target Concept Bridge Concept Value	Remove Concept Bridge
		Change Concept Bridge Concept Value (to super concept)
MERP2	How to handle invalid Source OR Target Property Bridge Argument Value	Remove Property Bridge
		Change Property Bridge Argument Value
MERP3	How to handle invalid generic/extensional specification condition	Remove Bridge
		Remove Invalid Condition
MERP4	How to handle orphaned Concept Bridge	Delete orphaned Concept Bridge
		Reconnect orphaned Concept Bridge to parent
		Keep orphaned Concept Bridge
MERP5	How to handle Property Bridges whose Concept Bridge was removed	Don't Propagate any Property Bridges (Remove Property Bridges)
		Propagate direct Property Bridges to sub Concept Bridges
		Propagate (All) Property Bridges to sub Concept Bridges

Table 15 – Mapping evolution resolution points and elementary evolution strategy

Both resolution point and evolution strategy are key concepts on the parameterization of the supported ontology mapping evolution process.

6.2 Mapping Change Resolution

This section explains the resolution of mapping changes that can cause ambiguities during the process according to the user parameterizations. Next sub-sections explain how the identified mapping changes, and consequent ambiguities, are solved.

6.2.1 Removing Concept Bridge

Let's analyze the case in which the user chooses the "Remove Concept Bridge" corrective action. Despite this option appears to be an easy way to correct the Concept Bridge, as analyzed in section 6.1.2.1 there are many implications in the ontology mapping document in order to solve inconsistencies.

The mandatory consequences are related to the internal content of the Concept Bridge:

- Remove Concept Bridge's Source Concept Value;
- Remove Concept Bridge's Target Concept Value;
- Remove Concept Bridge's Generic Conditions;
- Remove Concept Bridge's Extensional Specification.

Other consequences arise from the selected evolution strategy. In the case of the "Remove Concept Bridge", if the removed Concept Bridge has sub Concept Bridges and Property Bridges, there are two related mapping evolution resolution points to deal with:

- How to handle Orphaned Concept Bridges;
- How to handle Property Bridge propagation.

Accordingly, every evolution strategy of the first resolution point is followed by one of the three elementary evolution strategies of the second resolution point. In other words, these two mapping evolution resolution points are related, and the execution of the first one constrains the second one. Figure 52 illustrates the dependencies between them.

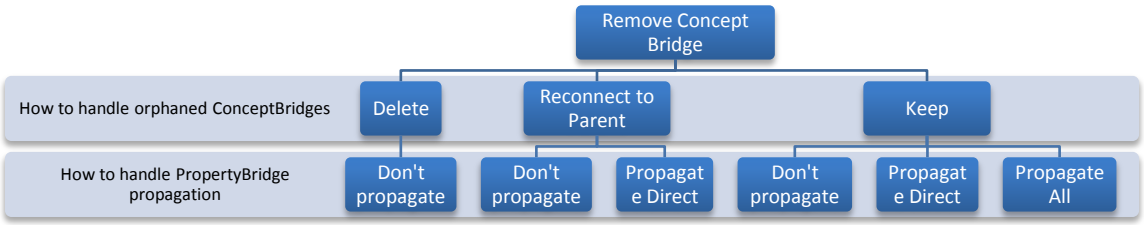

Figure 52 – Dependencies between mapping evolution strategies when remove Concept Bridge

Next sub-sections explain for each combination of the two resolution points, the change resolution process.

6.2.1.1 Delete Orphaned Concept Bridges and Don't Propagate

There is a set of necessary mapping changes to solve the inconsistencies in ontology mapping document. Figure 53 illustrates the main consequences of removing a Concept Bridge.

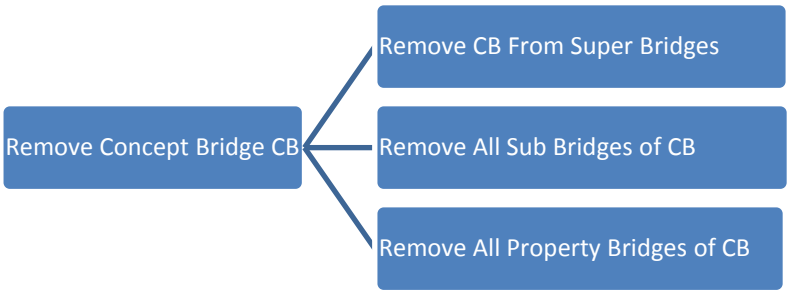

Figure 53 – Consequences of removing a Concept Bridge

Accordingly, when a Concept Bridge is removed, the following actions are taken:

- Remove all its subBridgeOf relations with all its super Concept Bridges;
- Remove all its sub Concept Bridges;
- Remove all its Property Bridges.

Algorithm 1 describes the implemented approach, implemented in the Concept Bridge class.

Algorithm 1: Delete Orphaned Concept Bridges and don't Propagate

```
1:  setSuperBridges = getSuperBridges();
2:  setSubBridges = getSubBridges();
3:  setPropertyBridges = getPropertyBridges();
4:  foreach superBridge in setSuperBridges
5:      removeFromSuperBridge(superBridge);
6:  endForeach
7:  foreach subBridge in setSubBridges
8:      subBridge.remove();
9:  endForeach
10: foreach propertyBridge in setPropertyBridges
11:     propertyBridge.remove();
12: endForeach
```

6.2.1.2 Remove Concept Bridge and Reconnect Orphaned Concept Bridge to Parent

In this case, when a Concept Bridge is removed, the subBridgeOf relation to all its parents should be removed, reconnect all sub Concept Bridges to all super Concept Bridges and resolve Property Bridge

propagation. Figure 54 illustrates the consequences of removing a Concept Bridge and reconnecting its orphaned sub Concept Bridges to their parents.

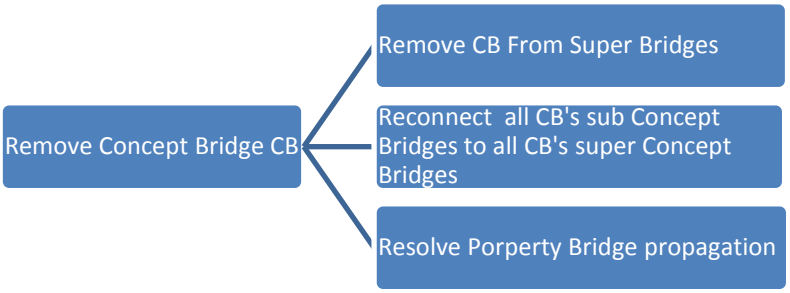

Figure 54 – Consequences of remove a Concept Bridge and reconnect sub Concept Bridges to its parent

Algorithm 2 describes the implemented approach to resolve a "remove Concept Bridge" change when the choice for the first resolution point is "reconnect orphaned Concept Bridge to parent". This method is implemented in the Concept Bridge class.

Algorithm 2: Reconnect Orphaned Concept Bridge to Parent

```
1:  setSuperBridges = getSuperBridges();
2:  setSubBridges = getSubBridges();
3:  setPropertyBridges = getPropertyBridges();
4:  foreach superBirdge in setSuperBridges
5:      removeFromSuperBridge(superBridge);
6:      foreach subBridge in setSubBridges
7:          subBridge.reconnectTo(superBridge);
8:      endForeach
9:  endForeach
10: handlePropertyBridgePropagation();
```

The "handlePropertyBridgePropagation" procedure is further described in Algorithm 4.

Notice that the "propagate all Property Bridges" option doesn't make sense in this context because Property Bridges from parents are (automatically) inherited by the sub Concept Bridges.

6.2.1.3 Keep Orphaned Concept Bridge

There are a set of necessary actions that are necessary in order to make ontology mapping document consistent when "Remove Concept Bridge and Keep Orphaned Concept Bridge". Figure 55 illustrates the main consequences for this case.

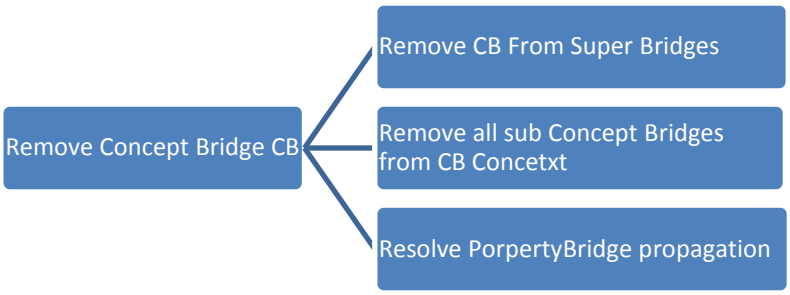

Figure 55 – Consequences of remove a Concept Bridge and keep sub Concept Bridges

In this case, when a Concept Bridge is removed, it is necessary to remove the subBridgeOf relation with all its parent and child, and resolve Property Bridge propagation. Algorithm 3 describes the implemented approach. The method is implemented in the Concept Bridge class.

Algorithm 3: Remove Concept Bridge and Keep Orphaned Concept Bridge

```
1:  setSuperBridges = getSuperBridges();
2:  setSubBridges = getSubBridges();
3:  setPropertyBridges = getPropertyBridges();
4:  foreach superBirdge in setSuperBridges
5:      removeFromSuperBridge(superBridge);
6:  endForeach
7:  foreach subBridge in setSubBridges
8:      subBridge.removeFromSuperBridge (this);
9:  endForeach
10: handlePropertyBridgePropagation();
```

Again, procedure “handlePropertyBridgePropagation” is further described in Algorithm 4.

6.2.1.4 Handle Property Bridge Propagation

The way to deal with Property Bridges depends on the evolution strategy chosen to handle with Property Bridge propagation and the evolution strategy chosen to handle with orphaned Concept Bridges.

This procedure is described in Algorithm 4, implemented in the Concept Bridge class.

Algorithm 4: handle Property Bridge Propagation

```
1:  Require: ES_PB = Evolution Strategy from Handle Property Bridge Propagation
2:  Require: ES_CB = Evolution Strategy from Handle Orphaned Concept Bridges
3:  if ES_PB is PROPAGATE_NO then
4:      setPropertyBridges = getPropertyBridges();
```

```
5:      foreach propertyBridge in setPropertyBridge
6:          propertyBridge.remove();
7:      endForeach
8:      return;
9:  endIf
10: if ES_PB is PROPAGATE_ALL and ES_CB is RECONNECT_TO_ROOT then
11:     setPropertyBridges = getALLPropertyBridges();
12: else
13:     setPropertyBridges. getPropertyBridges();
14: endif
15: foreach propertyBridge in setPropertyBridges
16:     if propertyBridge.isFrom(this) then
17:         propertyBridge.propagateTo(setSubBridges);
18:         propertyBridge.removeFrom(this);
19:     else //propertyBridge from parents
20:         propertyBridge.propagateTo(setSubBridge);
21: endif
```

6.2.2 Changing Concept Bridge Concept Value

This chapter analyzes the case in which the user chooses the "Change Concept Bridge Concept Value" corrective action. As analyzed in section 6.1.2.2, there are implications in the ontology mapping document in order to solve inconsistencies. If there are Property Bridges in which the paths cannot be updated according to the new concept value, there is an ambiguous situation during mapping change resolution. As consequence, "How to handle Property Bridge propagation" resolution point, suggests two alternative resolutions (Figure 56);

- Don't Propagate any Property Bridges (Remove Property Bridges);
- Propagate direct Property Bridges to sub Concept Bridges.

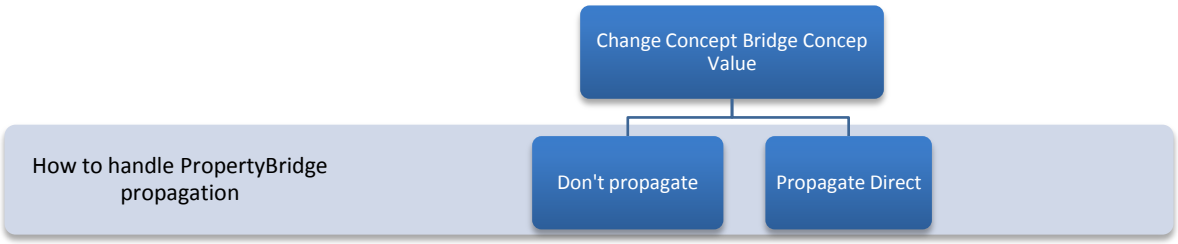

Figure 56 – Dependencies between mapping evolution strategies when Chance Concept Bridge Value

Next two sub-sections explain these two resolution processes.

6.2.2.1 Change Concept Value and Don't Propagate Property Bridges

There are a set of necessary mapping changes to perform in order to solve inconsistencies in ontology mapping document when "Changing Concept Bridge Value and Don't Propagate Property Bridges" occurs. Figure 57 illustrates the main consequences of this mapping change when the change occurs on the target concept value.

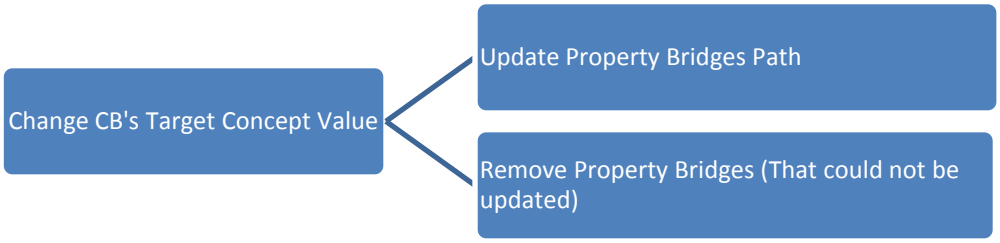

Figure 57 – Consequences of Change Concept Bridge Concept value and don't propagate

According to Figure 57, the following mapping changes are performed:

- Update Property Bridges paths according to the new concept value. This action allows the preservation of the semantics since the Property Bridges are already specified in the context of that Concept Bridge. However, this process is not possible when the property (path) existed only in the domain of the removed concept. The Property Bridges whose paths cannot be update remain inconsistent;
- Remove all inconsistent Property Bridges, i.e., remove all Property Bridges that could not be updated in previous step.

Algorithm 5 describes previous procedure in a more formal way. This method is implemented in Concept Bridge class.

Algorithm 5: Change Concept Bridge target Concept Value and don't propagate

```
1:  superBridge = getSuperBridge();
2:  setPropertyBridges = getPropertyBridges();
3:  targetSuperConcept = superBridge.getTargetConceptValue();
4:  this.setTargetConcetpValue(targetConcetpValue);
5:  foreach propertyBirdge in setPropertyBridges
6:      if not propertyBridge.updatePaths() then
7:          propertyBridge.remove();
8:      endif
9:  endForeach
```

6.2.2.2 Change Concept Value and Propagate Direct Property Bridges

When "Changing Concept Bridge Value and Propagate Direct Property Bridges" occurs, there are a set of necessary mapping changes to perform in order to solve inconsistencies in ontology mapping document.

Figure 58 illustrates the main consequences of this mapping change when the change occurs on the target concept value.

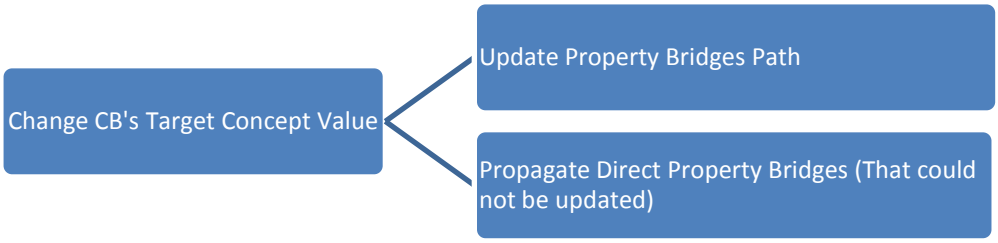

Figure 58 – Consequences of Change Concept Bridge Concept value and Propagate Direct

The process can be informally described as follows:

- Update Property Bridges paths according to the new concept value. Again, this action allows the preservation of the semantics since the Property Bridges are already specified in the context of that Concept Bridge. However, this process is not possible when the property (path) existed only in the domain of the removed concept. The Property Bridges that the path cannot be update remains inconsistent;
- Propagate all inconsistent Property Bridges (the Property Bridge that could not be updated) to sub the Concept Bridges.

Algorithm 6 describes previous procedure in a more formal way (Method in Class Concept Bridge).

Algorithm 6: Change Concept Bridge target Concept Value and propagate direct Property Bridges

```
1:  superBridge = getSuperBridge();
2:  setSubBridges = getSubBridges();
3:  setPropertyBridges = getPropertyBridges();
4:  targetSuperConcept = superBridge.getTargetConceptValue();
5:  setTargetConcetpValue(targetConcetpValue);
6:  foreach propertyBirdge in setPropertyBridges
7:      if not propertyBridge.updatePaths() then
8:          propertyBridge.propagateTo(setSubBridges);
9:      endif
10: endForeach
```

6.3 Conclusion

This chapter presented an approach to correct invalid SBO entities without ontology evolution information. The proposed approach suggests a semi-automatic process where the user parameterizes the system before hand, so the process runs autonomously thereafter. The parameters are based in a concept of mapping evolution strategy developed in this work. Some open issues and limitations are described in section 7.2.

Despite such issues, the proposed approach allows the evolution of the ontology mapping document, and sometimes, depending on the user parameters, it can achieve the same results of the process that uses the information provided by ontology evolution phase.

Chapter 7

Conclusion

The development of this work helped to acquire a deeper knowledge about the subject. In this chapter a backward analysis and forward prospect is presented. Next sub-section summarizes the key achievements of this work (section 7.1), show the remaining open problems and limitations and proposes some light on how to solve them (section 7.2).

7.1 Outlook of This Work

The work described in this thesis has the following main parts:

- State of the art;
- Conceptual achievements;
- Technological achievements.

7.1.1 State of The Art

- Ontologies and Ontology mapping process was presented, highlighting the most common definitions, application areas and representation languages, the concept ontology mapping and the implemented approach in MAFRA Toolkit.

- KAON Ontology Evolution Approach – The concept of ontology evolution strategy proposed in this approach was very useful for this work, and was adapted for ontology mapping evolution. Through the identified ontology evolution scenarios, the resulting information is used to improve ontology mapping evolution process;
- SIKS Ontology Evolution Approach – Centered on ontology changes according to Gruber's definition of ontology [Gruber, 1993]: **specification** of a **conceptualization** which is usually **represented** in some ontology language. Besides this three (independent) level presented in this approach, this work is centered on **Specification Change**. **Changes in conceptualization** mean changes in the meaning of ontology entities. Some of these changes (e.g. Concept *Jaguar* that previously means jaguar the car, changes to *Jaguar* the Animal) which don't reflect in the others level are very complex to capture and rarely there are (meta) information about them. **Changes in representation** means changes in how ontology is serialized, and ontology mapping document in independent from that. The first horizontal layer of MAFRA architecture (Lift & Normalization) is responsible for raising all data to be mapped to the same representation level. This work deals with changes at conceptual level that are reflected at specification level;
- WISE Ontology Evolution Approach – Proposes a five phase's process, quite similar to KAON ontology evolution approach. Beyond these five phases, this approach has a key element named Version log which contains a list of the different version of ontology concepts and it is used to serve as the base for definition changes and as source for change detection. This approach also provides a log that keeps track of all applied changes and gives an overview of the complete evolution history of the ontology in terms of changes.

7.1.2 Conceptual Achievements

- Ontology Evolution Vs. Ontology Mapping evolution, respects the analysis and systematization of the ontology evolution process in order to provide insights to the ontology mapping evolution process. In other words, understanding how ontologies evolve and devise the reasons (i.e. the semantics) they convey, such this information can be exploited during the ontology mapping evolution;
- How ontology mapping changes are represented in a formal way, how they are related and how they should be executed to keep ontology mapping document consistent:
  - A set of ontology mapping changes was developed, based on SBO (Semantic Bridge Ontology) language;
  - A change dependency matrix was created, containing a list of consequent changes for each change;

- Ontology changes representation, understand how ontology changes are classified and formally represented, was crucial since main goal of the work is to adapt ontology mapping document according to ontology changes;
- Changes storage, beyond understand how ontology changes are classified and formally represented, it was very important to this work understands how changes are stored. All approaches present a concept of evolution log that tracks all changes performed in ontology;
- Correcting Invalid SBO Entities. This phase corrects all invalid SBO entity. This process comprises the identification of the invalid SBO entity, the reason why it is invalid, and depending on that, find a way to correct it. This work uses two distinct approaches to correct SBO entities:
    - Correcting with ontology evolution information from ontology evolution phase – This step uses ontology evolution scenarios provided from ontology evolution phase and adapt it to mapping evolution scenarios;
    - Correcting without ontology evolution information from ontology evolution phase – In this step for each reason that made the SBO entity invalid, there a configurable set of available ontology mapping change to correct it.

7.1.3 Technological Achievements

Parallel to the conceptual achievements described before, extensive effort has been putted in the development of tools that reflect the proposed ideas. In that context, several technological challenges were faced:

- **MAFRA Toolkit workflow** – Since all work runs in the context of MAFRA Toolkit, understanding its core processes and flows was an important task. Moreover, due the complexity and dimension of the application, understanding the architecture and its implementation was a very difficult and time consuming process;
- **Semantic Bridging Ontology** – SBO (Semantic Bridging Ontology) is the ontology that specifies the entities used in ontology mapping document. In other words, an ontology mapping document is an instantiation of SBO. Understanding its semantics (i.e. SBO entities, the relation between them and their restrictions) and their application in scope of MAFRA Toolkit was a very important and time-consuming task;
- **KAON API for Ontology Evolution** – This API was used to access the evolution log (log that tracks all changes performed during ontology evolution process). Like all API or new tool, there are always difficulties to understand how it works. This is always a boring and time consuming task;
- **Correcting Invalid SBO entities** – This is one of the most important part of the technological perspective. Identifying and understanding why an SBO entity is invalid, what and how to correct it programmatically became the largest challenge faced during this project.

7.2 Open Issues and Future Work

This section identifies some open problems and limitations of the work described, proposing resolution approaches for each.

7.2.1 KAON Log Dependency

In the implemented approach there is a KAON Log dependency. In other words, the implemented approach reads kaon ontology evolution logs only. However there are many ontology management systems that provide ontology evolution logs containing useful information to use in ontology mapping evolution process. To use that information, an abstraction layer between the specific tools logs and the evolution information layer might be part of the solution. This layer is responsible for the abstraction of the differences between logs, providing to the evolution information log, a common representation of logs (Figure 59).

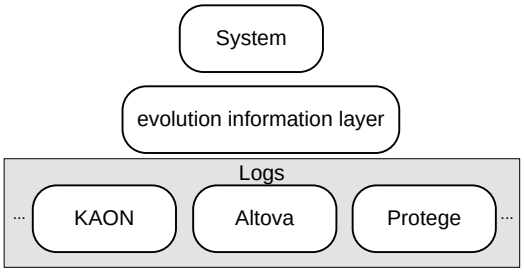

Figure 59 – Evolution information layer

7.2.2 No Semantic Guaranties for Some Actions

Some resolution changes are not semantically complete, as they require semantic information that is unavailable from the ontology evolution information.

In order to better elucidate this concept, let's consider the following example (Figure 60). Changing the source concept value to a super concept leads to a semantically invalid situation detected at transformation time only. At data transformation time, ambiguity arises, preventing the execution of one of the two Concept Bridges. This situation is referred as the extensional specification problem [Silva, 2004].

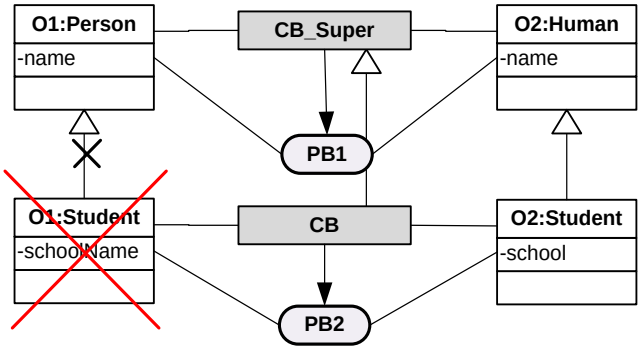

Figure 60 – Example No semantic guaranties (changing concept value)

In this scenario, the *CB* Concept Bridge is invalid because its source concept value does not exist anymore. Taking the "change concept value to super concept" action, *CB* will be structurally valid relating *O1:*Person to *O2:*Student. Although, the simultaneous existence of *CB_Super* and *CB* is ambiguous at transformation time (transforms instances of *O1:*Person in *O2:*Human or *O2:*Student).

In order to disambiguate this situation, an extensional specification condition should be added to one of the Concept Bridges. This process requires semantic considerations, normally unavailable from ontologies only.

Two mutually exclusive actions are suggested for solving this semantic ambiguity:

- Removing one of each ambiguous pair of entities. Considering the example of Figure 60, either *CB_Super* or *CB* would be removed. Obviously, apply this approach more information in mapping documents is probably lost, hence this is probably not the best approach;
- Exploiting ontologies' description logic axioms in order to discover extensional specification conditions. Imagine that Concept *O1:*Student is defined as subclass of *O1:*Person based on the "studiesAtSchool" property value. I.e. a *O1:*Student is a *O1:*Person that studies at a School. Therefore, the idea is to exploiting this axiom as semantic insight for defining the extensional specification condition to disambiguating both Concept Bridges.

7.2.3 Processing New Ontology Entities

This process has the goal to discover new semantic relations between new ontology entities and was not completely implemented since it is an adaptation of the ontology mapping process. Besides, it was not the purpose of this work.

In fact, ontology evolution process is not just removing entities. New ontology entities may be added to ontologies too. For these new entities, a (semi) automatic ontology mapping process is needed, calculating the similarity between entities and (if necessary) establish semantic relation between them.

After analyzing MAFRA Toolkit approach for (semi) automatic ontology mapping process, two important steps were identified:

- **Similarity measure** aims to discover and measure similarities between source ontology entities and target ontology entities. This process is performed by matchers (Matching Algorithm Systems), which evaluate the similarity between pairs of source and target ontology entities. For this phase, depending on computational resources, ontology domains, the accuracy needed etc, the following trends are proposed:
  - **Level 1** – Measure similarity between new entities only;
  - **Level 2** – Measure similarity between new entities and all not bridged entities;
  - **Level 3** – Measure similarity between new entities and all entities;
  - **Level 4** – Measure similarity between all entities (pure ontology mapping process).
- **Automatic semantic bridging** transforms the matches provided by similarity measure phase into a valid ontology mapping document defined according to Semantic Bridge Ontology (SBO). For this phase, depending on the level chosen in previous phase, the semantic relations to establish varies. Beside this, some options are available:
  - **Replace semantic relations** – After discover new similarity, establishing new semantic relations may be more correct than existing ones. The replace or not replace the semantic relations;
  - **Change semantic relations arguments** – Besides replace semantic relations, depending on service applied on Property Bridge (1:n or n:1 cardinality of service), new ontology entity can be added to source or target Property Bridge argument.

7.2.4 Use Meta-information from Ontology Evolution

Semantic information from ontology evolution process is seldom provided by the user, but if available it tends to better describe the meaning of entities. This information is not currently used in this work as it was decided to do not address it. Yet, in a future stage, specific methods and process might be developed in order to apply such information for improvement and disambiguation of the ontology mapping evolution.

According to [Klein, 2004] there are three levels of ontology changes: Changes in conceptualization, specification and representation. This work deals with changes in specification (Concept definition and relations). Changes in representation (concept definition in a language) are not important since one layer of MAFRA architecture is to raise all data to de same level of representation. Some changes in conceptualization (human construct) may not reflect in specification level, but they need to be considered.

Some meta-information (e.g. comments, labels) is provided to help on definition of the real meaning of ontology terms. This kind of information should be used in ontology mapping evolution process.

7.2.5 Use Notion of Composite/Complex Ontology Changes

Changes with coarser granularity, like group or split concepts are not exploited in this work. Despite these kind of changes are rarely used, its support would be welcome.

For that, the ontology mapping process should use that information to improve the process. Use composite changes (e.g. pull concept up, split concept, group concept and merge concepts) like the specified in Table 2 is useful, allowing new ontology evolution scenarios that can be used to improve ontology mapping evolution process.

7.2.6 Take in Account Others Ontology Languages

This work deal with ontology changes in RDFS. MAFRA Toolkit normalizes all ontologies language to RDFS language. The Semantic Bridge Ontology used in MAFRA Toolkit was designed to meet this ontology language requirement only. Yet, as OWL (especially OWL-DL) is becoming more and more used, exploiting the expressivity features of OWL-DL is recommended.

To solve this, many adaptations are required to be done in MAFRA Toolkit implementation and in process to ontology mapping evolution process:

- **Lift & Normalization** – This layer of MAFRA Toolkit actually raises all ontology to RDFS language. If ontologies are in OWL language, the conversion between OWL to RDFS may lose information and semantics, since OWL can be considered a superset of RDFS. The normalization of ontology should be to OWL language in order to prevent lost of information and semantics;
- **Semantic Bridge Ontology** – SBO language is too object oriented and doesn't take advantage of some characteristics of OWL language (e.g. description logic);
- **Use description logic to improve mapping evolution process** – Actually, since the ontology language used is RDFS, the ontology mapping process doesn't take advantage of description logic. Using that, new scenario could be created to improve the process.

7.2.7 Evaluation

Work involving semantics is always difficult to deal with and to evaluate their results. Projects on similar subjects do not present any experiments or evaluations either. In fact, [Stojanovic, 2004] work suggests that when dealing with user defined strategies, the results are biased and should not be evaluated by experience or test cases since the results is subjective.

In that sense a mathematical formal approach might be followed, though a considerable different (or at least complementary) research direction should be considered.

7.3 Final Remarks

The most important benefits of this work relate to the acquisition of new and deeper knowledge about ontologies, ontology evolution process, ontology engineering, ontology mapping and respective evolutions. As consequence, besides the achieved results, the development of this work allowed expanding knowledge and gain awareness of complex and unexpected problems on this subject, which sight new challenges for the future.

References

Banerjee, Jay, et al. 1987. Semantics and implementation of schema evolution in object-oriented databases. *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data.* San Francisco, California, United States : ACM SIGMOD, 1987, Vol. 16, pp. 311-322.

Berners-Lee, Tim. 1999. *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor.* 1999.

Berners-Lee, Tim, Hendler, James and Lassila, Ora. 2001. The Semantic Web. *Scientific American.* 2001, Vol. 284, pp. 34-43.

Bontcheva, Kalina, et al. 2004. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering.* 2004, Vol. 10, pp. 349-373.

Brost, Willem Nico. 1997. *Constroction of Engineering Ontologies for knowledge Sharing and Reuse.* 1997. PhD Thesis.

Euzenat, Jérôme and Shvaiko, Pavel. 2007. *Ontology Matching.* Berlin : Springer-Verlag Berlin Heidelberg 2007, 2007.

Fensel, Dieter. 2001. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce.* Heidelberg, Germany : Springer-Verlag, 2001.

Gabel, Thomas, Sure, York and Voelker, Johanna. 2004. *KAON – An Overview.* 2004.

Gruber, T. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition.* 1993, Vol. 2, pp. 199-220.

IEEE 90, Institute of Electrical and Electronics Engineers. 1990. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York : s.n., 1990.

Klein, Michel. 2004. *Change Management for Distributed Ontologies.* 2004. PhD Thesis.

Madhavan, Jayant, Bernstein, Philip A. and Rahm, Erhard. 2001. Generic Schema Matching with Cupid. *In The VLDB Journal.* Roma : s.n., 2001.

Maedche, Alexander, et al. 2003. An infrastructure for searching, reusing and evolving distributed ontologies. *WWW '03: Proceedings of the 12th international conference on World Wide Web.* Budapest, Hungary : s.n., 2003, pp. 439-448.

Maedche, Alexander, et al. 2002. MAFRA - A Mapping Framework for Distributed Ontologies in the Semantic Web. *Proceedings of the ECAI Workshop on Knowledge Transformation.* July 2002.

Plessers, P., De Troyer, O. 2005. Ontology Change Detection using a Version. *The Semantic Web – ISWC 2005.* s.l. : Springer Berlin / Heidelberg, 2005.

Rahm, Erhard and Bernstein, Philip A. 2001. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases.* s.l. : Springer-Verlag 2001, 2001.

Schulten, Ying Ding, et al. 2003. The role of ontologies in eCommerce. [book auth.] Steffen Staab and Rudi Studer. *Handbook on Ontologies.* s.l. : Springer, 2003.

Silva, Nuno Alexandre Pinto. 2004. *Multi-Dimensional Service-Oriented Ontology Mapping.* 2004. PhD Thesis.

Stojanovic, Ljiljana. 2004. *Evolution, Methods and Tools for Ontology.* 2004. PhD Thesis.

Studer, Rudi, V Richard, Benjamins and Fensel, Dieter. 1998. Knowledge engineering: principles and methods. *Data Knowledge Engineering.* 1998, Vol. 25, pp. 161-197.

W3C/OWL. 2004. [Online] February 10, 2004. [Cited: June 26, 2008.] http://www.w3.org/TR/owl-features/.

W3C/RDF. 2004. [Online] February 10, 2004. [Cited: June 26, 2008.] http://www.w3.org/RDF/.

W3C/RDFS. 2004. [Online] February 10, 2004. [Cited: June 26, 2008.] http://www.w3.org/TR/rdf-schema/.

Wikipedia/SemanticWeb. 2008. [Online] July 1, 2008. [Cited: July 1, 2008.] http://en.wikipedia.org/wiki/Semantic_Web.