NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — "Semantic-based knowledge and content systems"

D1.1.6 KANNEL: a framework for detecting and managing semantic relations between ontologies

Deliverable Co-ordinator: **Carlo Allocca**

Deliverable Co-ordinating Institution: **Open University**

Other Authors: **Mathieu d'Aquin (Open University), Enrico Motta (Open University)**

In Semantic Web search engines such as Watson, or in large ontology repositories, we need to make explicit existing, implicit relations between ontologies in order to better support users and applications in selecting the "best" or the "right" ontologies according to their goals. We describe KANNEL, a framework for detecting and managing ontology relations in large ontology repositories. KANNEL relies on an ontology based approach. A complex ontology of relationships between ontologies is first formalized, and specialized detection mechanisms are devised for extracting particular relations from the repository. The ontology can then be used with reasoning mechanisms to query ontology relations and infer new ones.

Document Identifier:	NEON/2009/D1.1.6/v0.5	Date due:	October 31st, 2009
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	October 31st, 2009
Project start date	March 1, 2006	Version:	v0.5
Project duration:	4 years	State:	Final Version
		Distribution:	Public

NeOn Consortium

This document is part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 11 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SL–1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier, France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield, United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

- Open University
- Jožef Stefan Institute
- iSoCo
- Universität Karlsruhe
- INRIA
- Ontoprise
- CNR
- Universidad Politécnica de Madrid

Change Log

Version	Date	Amended by	Changes
0.1	19-07-2009	Carlo Allocca	Initial Version
0.2	28-09-2009	Carlo Allocca	update with ontology description, similarity and version
0.3	06-11-2009	Carlo Allocca	update with other detection mechanisms and evaluations
0.4	09-11-2009	Mathieu d'Aquin	Future work and final editing
0.5	09-12-2009	Carlo Allocca	update with the first review comments
0.5	13-12-2009	Mathieu d'Aquin	general review and comments
0.5	14-12-2009	Carlo Allocca	update and final editing

Executive Summary

Nowadays, more and more ontologies are collected and indexed by Semantic Web Search Engines (SWSEs), such as Watson. These ontologies are not necessarily stand-alone artifacts, i.e. semantically independent from each other. In fact, It is easy to find ontologies connected to each other through semantic relations, such as: *version*, *inclusion*, *similarity*, *equivalent*, etc. In this context, one of the most important challenges, which has not been addressed yet, is that the relations among these ontologies often remain implicit. The implicit nature of these relations leads to several problems. One of them is how to select the "best/right" ontology/ies for our application, automatically. Consequently, a flat list of results from a SWSE may be difficult to exploit for those applications which rely on the selection of a relevant ontology, using infrastructures like Watson. In order to use such SWSEs efficiently, these relations must become explicit.

Motivated by this concrete scenario, in this deliverable we present KANNEL: a framework to detect and manage existing relations automatically between ontologies in large ontology repositories. First of all, we present the ontology-based architecture of KANNEL. We also detail the design of DOOR - the Descriptive Ontology of Ontology Relations - which intends to formalize ontology relations such as *inclusion*, *versioning*, *similarity* and *agreement* using ontological primitives as well as rules. We finally introduce the methods to detect such relations in large ontology repositories and initial evaluations of these mechamisms.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem

The general problem we intend to address in this research, can be formulated by the following question:

(1) How can we (automatically) select the "right" or the "best" ontologies for our goal on the Semantic Web?

More and more ontologies are collected and indexed by Semantic Web Search Engines (SWSEs), such as Watson[1]. Indeed, SWSE systems represent one of the most important steps to reuse existing ontologies and play an important part of the Semantic Web (SW) [ABS06]. In general, SWSEs provide searching mechanisms and appear like traditional Search Engines to the users. When you type keywords, a flat list of documents, in which ontologies are stored, come out as result. However, there are significant differences between a generic Search Engine, such as Google, Yahoo and a generic SWSE, such as Watson. The former collects and indexes any type of documents, while the latter only manages semantic documents[2]. Search Engines' output are supposed to be used by humans, while SWSEs' output are used by Semantic Web Applications (SWAs) (in fact, more and more real-world SWAs use the Semantic Web (SW) as a large-scale knowledge source [dMea08]). Another important difference is that semantic documents are not stand-alone artifacts, i.e., semantically independent from each other. They are related by (often implicit) semantic relations. Not all of these differences are taken in consideration when SWSEs are developed. In particular, in this deliverable we cope with just the last aspect: what are the relations between ontologies and how to automatically detect them in large ontology repositories. Thus, focusing on SWSEs the question (1) can be reformulated as the following:

(2) How can we improve the selection of ontologies beyond the flat list of results of SWSEs?

In question (2) there are two important aspects which need to be clarified. The first one concerns the task of "selection of ontologies". The authors of [MSM06] formalized and discussed the problem of selecting ontologies from the Semantic Web. The authors, on the one hand, formulated some relevant requirements, such as *complete coverage*, *Precise coverage*, *Returning ontology combinations*, *Dealing with relations*, *Dealing with instances*, etc, should be considered to design an algorithm for the selection of ontologies. On the other hand, they also agree on the fact that different applications have different requirements for ontology selection. With respect to our problem, reformulated with the question (2), we consider a different requirement: the *Connectedness*. The selection of an ontologies takes into consideration/account how the ontologies are related (or connected) to each other on the Semantic Web.

The second aspect of (2) which needs to be clarified is concerned with how the selection task can be improved taking into consideration the relations between ontologies, or in other words the factor of *Connectedness*. The way of linking the first with the second can be described by considering the following scenarios

[1] `http://Watson.kmi.open.ac.uk`

[2] In this work, a semantic document is the document in which is kept an artifact semantic such as the ontologies or knowledge expressed by any one of languages used for the Semantic Web, such as RDF, OWL.

that we intend to cope with in this work: we want to select on the Semantic Web an ontology that satisfies one or more of the following:

- selecting the **largest** ontology to cover the domain of interest;
- selecting an ontology which is **compatible** with other ontologies that are used by the current system;
- selecting an ontology which is **similar** to the one that the system is currently using;
- selecting the **smallest** ontology to cover the domain of interest;
- selecting the **last version** of the ontology that I am looking for;

All of the keywords such as *largest*, *compatible*, *similar*, *last version* etc, are related to the general notion of *Connectedness*, that is to the relations between ontologies. This is one of the most important aspects which drove the development of the KANNEL framework (Section 1.3).

On the same line of research, it is worth noting the work of [Wel08]. They identified the following questions related to the ontology reuse: how can the ontologies be organized or classified? should they be interrelated? and how can they be reused? These questions affect both the processes of ontology development and ontology applications. Furthermore, according to the author of [Wel08] we also need a more comprehensive overview of the ontologies on the Semantic Web, to handle a growing number of single, originally independent ontologies.

1.2 Motivation

The ontologies on the Semantic Web are not isolated artifacts. More and more researchers faced the fact that they are, implicitly, related to each other by one or more relationships. Making explicit these relations is a very hard task and plays an important role in the contest of SWSEs in the task of ontology selection.
The relevance to study relationship between ontologies can be discussed either at both practical and theoretical level [KA05].
At a theoretical level, studies have targeted ontology comparison in order to identify overlaps between them [MS02]. Many approaches have been proposed to find differences between versions of ontologies [NM02, KCDF08]. According to [KF01], the ontology versioning problem has been defined as *the ability to handle changes in ontologies by creating and managing different variants of it.* In other words, ontology versioning means that there are multiple variants of an ontology. The authors of [KF01] suggested that, ideally, developers should maintain not only the different versions of an ontology, but also some information about the way versions differ and whether or not they are compatible with each other. In [GPS99] ontology integration is defined as the construction of an ontology C that formally specifies the union of the vocabularies of two other ontologies A and B. The most interesting case is when A and B are supposed to commit to the conceptualization of the same domain of interest or of two overlapping domains. In particular, A and B may be related by being *alternative ontologies*, *truly overlapping ontologies*, *equivalent ontologies with vocabulary mismatches*, *overlapping ontologies with disjoint domain*, *homonymically overlapping ontologies*. Finally, in ontology matching, an alignment is a set of correspondences between the entities of two ontologies, therefore relating these two ontologies by mapping their models with each other.
At a practical level, we extend what we said in the previous section, showing more examples of how the ontologies are related to each other on the SW.
The SWSEs provide keyword based search mechanisms to locate relevant ontologies for particular applications. As an example, if we type the query "*student*", Watson [3]currently, gives 1079 ontologies as result (valid on the 22/04/2009), which are provided as a simple list. Indeed, on the first few pages we can observe some common situation which need to be investigated:

[3] `http://Watson.kmi.open.ac.uk`

1. when an ontology has been translated in different ontology languages:
   - `http://reliant.teknowledge.com/DAML/Mid-level-ontology.owl`
   - `http://reliant.teknowledge.com/DAML/Mid-level-ontology.daml`
2. different versions of the same ontology:
   - `http://www.vistology.com/ont/tests/student1.owl`
   - `http://www.vistology.com/ont/tests/student2.owl`
   - `http://lsdis.cs.uga.edu/projects/semdis/sweto/testbedv1.1.owl`
   - `http://lsdis.cs.uga.edu/projects/semdis/sweto/testbedv1.4.owl`

Inspecting more results of Watson in the same way, it is possible to find ontologies connected to each other, more sophisticated semantic implicit relations, such as *compatibility*, *inclusion*, *similarity*, etc.

On the other hand, Semantic Web Applications use the SW as a large-scale knowledge source [dMea08]: they achieve their tasks by automatically retrieving and exploiting knowledge from the SW as a whole, using advanced Semantic Web Search Engines (SWSEs) such as Watson [dSD$^+$07]. Therefore, these new applications explore the Web to discover ontologies relevant to the task at hand, thanks to SWSEs. The lack of semantic relations between ontologies in SWSEs leads to several inadequacies in supporting the development of SWAs. In the particular case of Watson, it generates additional difficulties in exploiting the provided result in terms of:

1. *Searching/Selecting appropriate ontologies*: Applications and human users need to select appropriate ontologies, from Watson's result, on the basis of some requirements. This aspect is not tackled by current SWSEs.
   - **User:** For example, if the user is looking for the last version of a particular ontology O, having a simple list as result from Watson means that the user needs to manually go through each ontology and find out which one is the last version. The same applies for other implicit semantic relations such as equivalence, similarity, incompatibility.
   - **Applications:** The situation becomes even worse for applications, because they have to navigate through the set of results to find out which ontologies are appropriate to their tasks. Making explicit implicit semantic relations among ontologies gives a possibility to exploit this network of ontologies for applications to select the most relevant ontologies, and query for related ones.
   - **Redundancy of Result**: Getting a set of results with redundancy simply makes the result bigger and not better in terms of knowledge. In other words, it means the human users and the applications have to inspect more ontologies than needed.
2. *Combining suitable ontologies*: Given the distributed nature of the SW, developers cannot expect one unique ontology source to provide all the required knowledge for a given application. Therefore a typical Semantic Web application must select and integrate particular ontologies. In these cases, the result of integration needs to be consistent in order to be used by the application.

In conclusion, we are strongly motivated to investigate our problem (Section 1.1) by all these practical aspects, with the main aims to guarantee an efficient use of SWSEs, in particular of Watson, in ontology selection task, which is very important for supporting the development of the SWAs.

1.3 Goal

Supported by the previous motivations (Section 1.2), the problem discussed in Section 1.1 need to be addressed efficiently. Both the theoretical and practical needs formalizing explicit relations between ontologies

require a general study of these relations, providing a formal base for defining, manipulating and reasoning upon the links that relate ontologies online. To achieve this, we present KANNEL - an ontology based framework for detecting and managing ontology relations for a large ontology repository, such as Watson. The general idea of KANNEL is shown in the Figure 1.1.

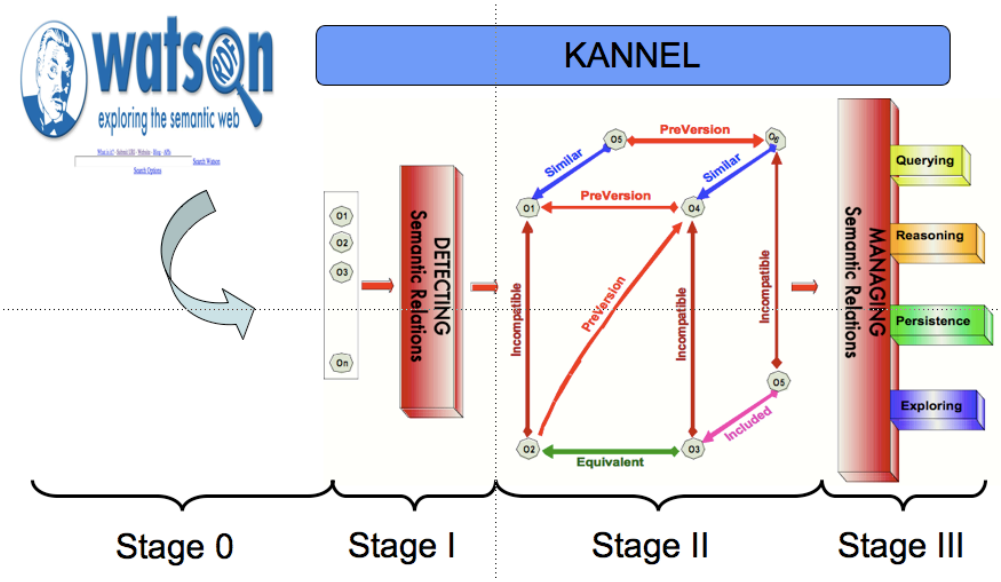

Figure 1.1: **The main tasks and stages of KANNEL.**

The process of discovering the underlying structure of the online ontologies to build a Network of Ontologies as shown in Figure 1.1 includes the following steps:

1. **Stage 0** concerns monitoring the Semantic Web to gather ontologies that are published and available on-line. This may be done by Watson, which collects and indexes ontologies and provides both keywords search based and complex formal query mechanisms. Thus, thank to Watson, we are able to reuse on-line ontologies or semantic data for our purpose. Watson produces a flat list of ontology URIs as output. This list represents the input for KANNEL. In general, Watson's ontology repository will be the input to the KANNEL framework.

2. **Stage I** Detecting Semantic Relations. It concerns the discovery and detection of existing semantic relations between ontologies. This stage is needed for the particular purpose of building the networked ontologies. At this stage it is very important to have a clear understanding of what relations between ontologies need to be detecting. Thus, specific detection mechanisms can be developed for the identified relations.

3. **Stage II** concerns building the ontology network from the detected relations (Stage I). It is a stage internal to KANNEL,which is in charge of keeping an appropriate semantic structure (DOOR, Chapter (3)) to hold all the links between the ontologies.

4. **Stage III** Managing Semantic Relations. Ones the ontology network has been built, we need to manage it and make it useful for the user and applications. For managing we mean keeping it up to date over the time and evolution of the Semantic Web. KANNEL provides an API and services such as 1) **Querying**: we are interested in expressing queries of the form *Which are the semantic relations between the ontologies* O_1 *and* O_2*?*, *Which are the ontologies related to the ontology* O_1*?* or *Which are the ontologies which have a vocabulary similar to the one of* O_1*?*; 2) **Reasoning**: we are interested in making reasoning among the ontology relations. For example, informally, if an ontology O_1 is included in the ontology O_2 and O_2 is included in O_1 then we can conclude that O_1 is equivalent to O_2, etc.

3)**Exploring/Visualization**: the need for the users to go through Watson's results, can be reformulated as finding the best way to display networks of ontologies, giving the possibility to navigate or explore the ontologies on the Semantic Web in a structured way, e.g. with respect to the satisfied relations among the ontologies.

1.4 Outcomes

Establishing semantic relations between ontologies represents a significative effort to improving SWSEs in ontology selection task. This will be demonstrated in the context of the Watson. KANNEL (Chapter 4), will provide efficient services and APIs to support both the SWAs and users in exploiting the Watson's result in terms of:

- *Searching/Selecting appropriates ontologies*: Explore and Navigate through the Watson's ontologies with respect to the semantic relations that are satisfied between them. By doing this, it makes easier the use of Watson for both the users and applications.
- *Combining suitable ontologies*: Making explicit implicit semantic relations to support the combining of ontologies consistently.
- *Looking at specific semantic relations:* Another advantage comes from the fact that we will be able to query the semantic relations which are satisfied among two or more ontologies. Semantic Relations among ontologies are then exploitable for themselves by applications.
- *Analytical level:* Acquire an overall understanding of the SW, its content, structure and evolution, facilitating the development of the SW in a controlled and informed way.

In general, the framework's services and APIs do not restrict the amount of ontologies that Watson provides but let the applications and users to consult them in an structured way, that is, with respect to the relations that link them.

1.5 Research Questions

To archive our goal (Section 1.3) the following research questions need to be answered:

1. **What are the semantic relations between ontologies?**

   We are interested in working out how the ontologies can be related to one another, in a context in which the SW is considered as a large-scale knowledge source. A first investigation of potential relations has led us to distinguish two sets of relations. The first set contains relations called *atomic*, which do not depend on others relations. An example of this type is the **inclusion** relation (Chapter 3). The second set, instead, contains relations called complex, which are defined using atomic relations or other *complex* relations. An example of this type is **equivalence** (Chapter 3) which can be defined in terms of inclusion one.
   Following this way of investigating, we break down the research question in two subquestions: **What is the set of atomic relations between ontologies?** and **What is the set of complex relations?**

2. **How can we manage the semantic relations between ontologies?** For this question, we need to investigate a number of subtasks of the management of semantic relations:

   (a) **How to represent semantic relations between ontologies?** By representation we mean the mechanisms needed to formalize or define semantic relations between ontologies. In addition, the concrete representation of semantic relations has an influence in the way they are stored.

(b) **How to reason about semantic relations between ontologies?** We intend to specify which kind of entailments we are interested in and which mechanisms we can use to compute them. A particular sort of reasoning can be shown by the following example: knowing that the ontology A is semantically equivalent to the ontology B and B is semantically incompatible with C, then A is incompatible with C. Having this relation explicit will help to avoid unexpected consequences to the applications that need to work with both ontologies A and C.

(c) **How to explore/navigate/interact with semantic relations between ontologies?** Changing the underlying structure of Watson's result into a graph, in which the set of nodes represent ontologies and the set of edges are the satisfied semantic relations between them, we need to come up with new ways to visualize the network of ontologies. We are investigating patterns of navigation/exploration for networks of ontologies.

To answer the research question (1) and (2.a, 2.b), we designed an appropriate methodology and developed DOOR (**D**escriptive **O**ntology of **O**ntology **R**elations). It is a semantic structure which formalize relevant relations between ontologies, Chapter 3.

3. **How to detect semantic relations between ontologies?** We are interested in methods for identifying/discovering semantic relations in large ontology repositories. In the (Chapter 4), ad-hoc ad-hoc mechanisms for detecting specific ontology relations are described.

1.6 Outline of this deliverable

This document is structured as follows: in Chapter 2 we continue discussing significant work concerning ontology relations; in Chapter 3 we present the adopted methodology to design DOOR (Descriptive Ontology of Ontology Relations) and we also describe the main parts of it. In Chapter 4 we describe the KANNEL framework. Finally, Chapter 5 concludes the document and sheds light on interesting future research of KANNEL.

Chapter 2

Related Work

Making explicit implicit relations between ontologies has brought us to investigate several branches of research including aspects such as ontology comparison (section 2.2) and ontology versioning (section 2.3). Before analyzing each related subject we discuss how ontologies can be related to each other (section 2.1).

2.1 How ontologies relate to one another?

The authors of [Wel05] have studied the question of how ontologies can be compared and integrated. They consider that the formalization of relations among ontologies is analogous to studying mathematical relations among the mathematical models of these ontologies. Thus, using the notion of model they characterize a number of relations among the ontologies[1]:

- *Ontologies representing the same conceptualization:*
    1. It means that the models of these ontologies are *equivalent* or
    2. It means that the models of these ontologies are *isomorphic*;
- *Resemblance between Ontologies:*
    1. It means that the models of these ontologies are *isomorphic*;
- *Simplification (Coarsening) of Ontologies:*
    1. It means that a model of the first ontology is a *homomorphic image*[2] of the second ontology;
- *Composition of Ontologies:* Complex domains include knowledge from other domains. Thus, complex domains could be described by the concepts related to other domains. Therefore, ontologies of complex domains are built from components, which are ontologies of other domains. In terms of model, every model of an ontology for a complex domain is the *product* of ontology models that are components.

Concluding, the authors proposed several relations between ontologies at a very abstract level, characterizing the relations among ontologies by relations corresponding to their models. Therefore, while interesting from a theoretical point of view, this work does not provide concrete elements for detecting and managing the considered relations. For this, we need to look at ontology comparison approaches.

[1] Please, refer to [Art] for complete definitions of the relations.

[2] An ontology O_2 will be called a homomorphic image of an ontology O_1 if there is a completely defined one-valued map h_1 from the set of model of O_1 to the set of model of O_2. Further, in this case we will say that there is a homomorphism $h_1 : O_1 \text{ -> } O_2$.

2.2 Ontology Comparison

- **What is it?** Comparing two or more ontologies is a way to find out whether and to what degree they overlap [MS02]. There are, in the literature, many approaches on how to compare two different versions of an ontology, in order to find the differences.
- **Why is it useful for us?** Providing different approaches to compare ontologies at different levels is relevant for our aims because the result of the comparison can be a relation among ontologies, stating for example that they are equivalent, or that one includes the other.
- **What is missing for us?** So far, the existing methods compare ontologies at a syntactical or structural level, while we are interested in comparing ontologies at a semantic level or conceptual level.

In the following we describe the most relevant works in this area. In particular we discuss the following two articles [NM02] and [KCDF08] in subsection 2.2.1 and 2.2.2 respectively.

2.2.1 PromptDiff: A fixed-point algorithm for comparing ontology versions

In the domain of Software Engineering, comparison of versions of the software code involve a comparison of text files and the result is a list of lines that differ in the two versions. This approach does not work for comparing ontologies because two ontologies can be exactly the same but have very different text representations. For example:

- their storage syntax may be different;
- the order in which definitions are introduced in the text file may be different;
- a representation language may have several mechanisms to express the same thing.

Trying to avoid the limitation of the previous method, *N. F. Noy and M. A. Musen* have proposed PROMPTDIFF.
The idea behind PROMPTDIFF is to compare the structure of ontology versions and not their text serialization. The PROMPTDIFF algorithm consists of two parts:

1. an extensible set of heuristic matchers; each matcher employs a small number of structural properties of the ontologies to produce matches;
2. a fixed-point algorithm to combine the results of the matchers to produce a structural diff between two versions;

The main concept behind PROMPTDIFF is a Diff structure which is defined as follows:

Definition 1 (*Diff Structure*) *Given two versions of an ontology O, V1 and V2, a structural diff between V1 and V2, D(V1, V2), is a set of frame pairs F1, F2 where:*

- *F1* $\in$ *V1 or F1 = null; F2* $\in$ *V2 or F2 = null;*
- *F2 is an image of F1 (matches F1), that is, F1 became F2. If F1 or F2 is null, then we say that F2 or F1 respectively does not have a match.*
- *Each frame from V1 and V2 appears in at least one pair.*

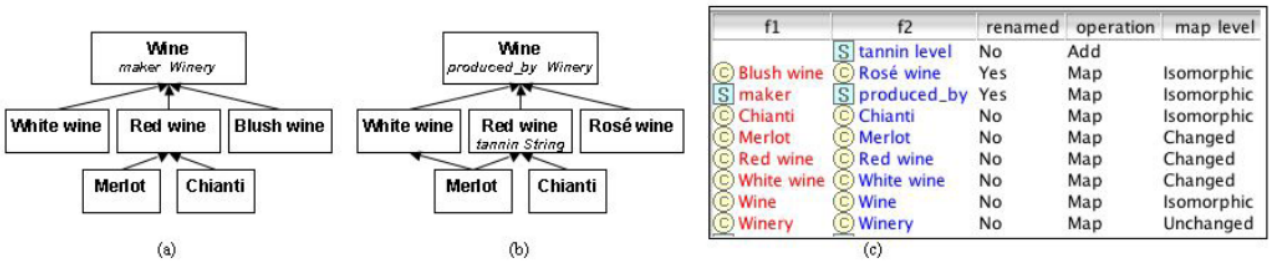

Figure 2.1: **An example of applying the Diff Structure**

- *For any frame F1, if there is at least one pair containing F1, where F2 6= null, then there is no pair containing F1 where F2 = null (if we found at least one match for F1, we do not have a pair that says that F1 is unmatched). The same is true for F2.*

An example of applying this definition is shown in Figure 2.1:
Suppose that we are developing an ontology of wine. In the first version (Figure 2.1a), there is a class Wine with three subclasses, Red wine, White wine, and Blush wine. The class Wine has a slot whose values are instances of the class Winery. The class Red wine has two subclasses, Chianti and Merlot. Figure 2.1b shows a later version of the same ontology fragment. Note the changes: we changed the name of the slot to produced by and the name of the Blush wine class to Rose' wine; we added a tannin level slot to the Red wine class; and we discovered that Merlot, can be white and added another superclass to the Merlot class. Figure 2.1c shows the differences between the two versions in a table produced automatically by PROMPTDIFF. Informally, given two versions of an ontology O, V1 and V2, two frames F1 fromV1 and F2 from V2 match if F1 became F2. This work provides an automatic way to compare different versions of the ontologies based on their structure in terms of their frame (class, instances, property, etc), describing not only what has changed but also showing to the user some information on how the frames have changed.
With respect to our objective, which is to detect semantic relations between ontologies, a structure comparison, on which PROMPTDIFF is based, is restrictive because the same conceptualization can be expressed by different axioms. For example, defining two ontologies O1 and O2 as O1 = sub(A,B), sub(B,C) and O2 = sub(A,B), sub(B,C), sub(A,C) . A structure comparison is not sufficient to recognize that the axiom sub(A,C) from O2 is already expressed through the axioms sub(A,B) and sub(B,C). Therefore, sub(A,C) could be deleted without losing any information from O2, and the comparison should conclude that O1 is equivalent to O2, as they have exactly the same models.

2.2.2 CEX and MEX: Logical Diff and Logic-based Module Extraction in a Fragment of OWL.

Recently, *B. Konev and his colleagues* have been developing **OWLDiff** (`http://semanticweb.org/wiki/OWLDiff`) which is a Pellet-backed tool that takes two OWL ontologies in input and computes their differences by entailment checking of the different axioms.
Actually, given two terminologies T_1 and T_2 over a signature Σ, OWLDiff, diff(T_1, T_2) calculates the differences as shown in Figure 2.2 and returns
a compact representation of diff(T_1, T_2). The motivation for developing OWLDiff emerged in the context of collaborative ontology authoring and sharing through SVN, as it became quite difficult to track changes in the same ontology made by different authors. This approach has been studied for the $\mathcal{EL}$ language, obtaining a polytime algorithm. However, ontologies that are commonly found online are expressed in formalisms closer to OWL-DL ($\mathcal{SHOIN}(\mathcal{D})$) than $\mathcal{EL}$, making OWLDiff unusable in practice for many cases.

$$\mathrm{diff}_{\Sigma}(T_1, T_2) = \left\{ C \sqsubseteq D \;\middle|\; \begin{array}{l} T_1 \not\models C \sqsubseteq D \\ T_2 \models C \sqsubseteq D \\ \mathrm{sig}(C \sqsubseteq D) \subseteq \Sigma \end{array} \right\}$$

Figure 2.2: **OWLDiff formula**

2.3 Ontology Versioning/Evolution

- **What is it?** The authors of [KF01] claim that there is no distinction between ontology evolution and ontology versioning and that ideally developers should maintain not only the different versions of an ontology, but also some information on how the versions differ and whether or not they are compatible with one another. Ontology versioning means that there are multiple variants of an ontology around. s often originates from changes made to an existing version of ontologies and thus form a derivation tree. In [KF01], the ontology versioning problem has been defined as *the ability to handle changes in ontologies by creating and managing different variants of it.*

- **Why is it useful for us?** The studies have been focusing on providing methodologies to recognize and manage different versions of an ontology [KF01].

- **What is missing for us?** The use of the provided methodologies are restricted to the ontology creation process. In our scenario (SWSEs), we need to recognize and manage the versioning relation when ontologies are made available online. Moreover, because of the distributive and dynamic nature of the Web it is likely that many *versions* and *variants* of ontologies are made available in an uncontrolled way. This can cause incompatibilities in the applications and ontologies that refer to them and may lead to wrong interpretations of the data [KF01].

In the following we describe the most relevant works in this area. In particular we discuss the following three articles [KFKO02], [KF01] and [**?**]. The first and second in subsection 2.3.1 and third in section 2.3.2.

2.3.1 Ontology Versioning

The authors *M. Klein and D. Fensel & others* addressed the following issues:

- Characterization of the ***version relation*** between ontologies;

- The identification of online ontologies;

- The design of a web based system that helps to keep track of different versions of the ontologies: OntoView;

Characterization of the *version relation* between ontologies

According to the authors of [KFKO02], there are three main aspects which need to be discussed when considering a version relation between ontologies.

1. about the difference between **conceptual relations inside an ontology** and **version relation**: the first one is a relation between concepts inside the ontology, it is important to model the domain of interest. The second one, instead, is a meta-relation to link ontologies to others. Considered properties of a *version relation* are:

   - transformation or actual change: a specification of what has been changed in an ontological definition, specified by a set of change operations (add class, remove property, etc)
   - conceptual relation: the relation between constructs in the two versions of the ontology, specified by an equivalence relation and a subsumption relation;
   - descriptive meta-data like **date, author and intention** of the update. This describes the when, who and why of the change.
   - scope: a description of the context in which the update is valid.

2. about the possible discrepancy between changes in the specification and changes to the conceptualization: a change in the specification does not necessarily imply a change in the conceptualization, because a language provides different mechanisms to express the same concept. Therefore, it is important to distinguish changes in ontologies that affect the conceptualization from changes that do not. The following terms are used to make this distinction:

   - **conceptual change**: a change in the way a domain is interpreted.
   - **explication change**: a change in the way the conceptualization is specified, without changing the conceptualization itself.

3. packaging of changes: it concerns the way in which updates are applied to an ontology. This aspect is important when you are building the ontology and not when it is already created.

The identification of online ontologies

The ontologies should have a unique and stable identification. According to the authors, in order to identify a version, the following questions must be answered:

1. what is the identity of an ontology?

2. How does this relate to web resources and their identity?

With respect to 1), the authors assume that an ontology is represented in a file on the web and every change that results in a different character representation of the ontology constitutes a revision. In case the logical definitions are not changed, it is the responsibility of the author of the revision to decide whether this revision is a conceptual change and thus forms an new conceptualization with its own identity, or just a change in the representation of the same conceptualization. At this point, it very important to take into account the conceptual and explication changes. With respect to 2), the ontology (as a set of concepts, relations and individuals) can be regarded as a resource and a URI can be used to identify it. Notice that URI's provide a general identification mechanism, as opposed to URL, which are bound to the location of a resource. In other words, a revision, which is normally specified in a new file, may constitute a new ontology, but this is not automatic. Every revision is a new file resource and gets a new file identifier,

but does not automatically get a new ontology identifier. If a change does not constitute a conceptual change, the new version gets a new location, but does not get a new identifier. For example, the location of an ontology can change from /example/1.0/rev0 to /example/1.0/rev1, while the identifier is still /example/1.0/.

Based on these ideas they have proposed an identification method that is based on the following points:

- a distinction between three classes of resources: 1. files; 2. ontologies; 3. lines of backward compatible ontologies.
- a change in a file results in a new file identifier;
- the use of a URL for the file identification;
- a change in the conceptualization or in the logical definition results in a new ontology identifier, but a non-logical explication change does not;
- a separate URI for ontology identification with a two level numbering scheme:
- individual concepts or relations, whose identifier only differs in minor number, are assumed to be equivalent;
- ontologies are referred to by an ontology URI according to major revision number and the minimal extra commitment, i.e., the lowest necessary minor revision number

Design of a web based system that helps to keep track of different versions of ontologies: OntoView Based on this theoretical aspects of ontology versioning, the authors describe a system, still under costruction, to help users to manage changes in ontologies and keep ontology versions as much interoperable as possible. It does this by comparing versions of ontologies and highlighting the differences. One of the central features of OntoView is comparing the ontologies at a structural level, showing which definitions of ontological concepts or properties changed. An example of such a graphical comparison of two versions of ontologies is shown in Figure 2.3
The comparison function distinguishes the following types of changes:

- Non-Logical changes;
- Logical Definition;
- Addition of definitions;
- Deletion of definitions;

First, as we can see from the figure, OntoView is able to compare two ontologies from a structural point of view. Second, their approach is useful in context where two ontologies are being created by different people in different places in that they could, for example, discover that they have the same conceptualization.

2.3.2 SemVersion, Versioning RDF and Ontologies

Another relevant work is presented in [Vol06] where the authors introduced **SemVersion**: it is a generic, extendable multi-language ontology versioning system. The design of SemVersion faces several questions, including **comparing versions** in terms of *How can changes between versions be computed? What is the right level of abstraction? The ability to handle arbitrary RDF requires to handling of blank nodes as well, which complicates diff computation.* According to the authors, a semantic versioning system needs the ability to compute a *diff*, which tells the user what (conceptually) has changed between two versions. In particular,

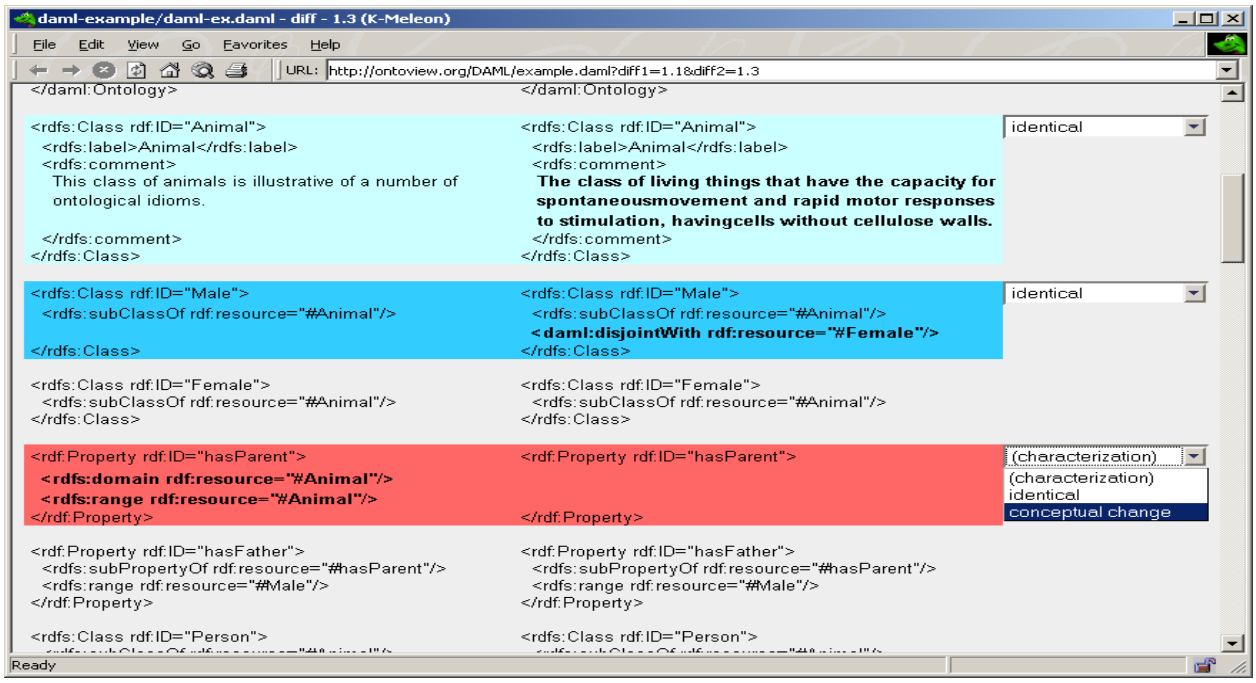

Figure 2.3: **- Screenshot of OntoView**

diff serves two purposes: First, SemVersion allows to compute (structural and semantic) diffs between two arbitrary chosen models, second to inform the user about changes. In SemVersion are distinguished three levels of diffs:*Set-Based Diff*, *Structural Diff* and *Semantic Diff*. Before going into the details of each of them, it is necessary to define a **diff function** as following: Let A and B be two versions of an ontology

Definition 2 (*diff function*) *The **diff function** d(A,B)*$\rightarrow < a(A,B), r(A,B) >$ *is a non commutative function from two triple sets (A, B) to two triple sets of added (a(A,B)) and removed (r(A,B)) statements, with*

- *a(A,B)= B-A;*
- *r(A,B)= A-B;*

In Figure 2.4 we report an example of the *diff function*

version A:	version B:
a rdfs:type c b rdfs:type c c rdfs:subClassOf d	a rdfs:type d b rdfs:type d c rdfs:subClassOf d e rdfs:type d
added $a(A,B)$:	removed $r(A,B)$:
a rdfs:type d b rdfs:type d e rdfs:type d	a rdfs:type c b rdfs:type c

Figure 2.4: **- Example A -**

- **Set-Based Diff**: Let A and B be two RDF triple sets. Computing *Set-Based Diff* between A and B means to calculate the set-theoretic difference between A and B, by simple set arithmetics for triple sets.

- **Structural Diff**: The structural diff is based on *Set-Based Diff*. In fact, in cases in which A and B do not contain blank nodes the structural diff reduces to a set-based diff. Otherwise, first, a particular extension of the models A and B is computed, introducing the *Blank Node Enrichment*, and then a set-based diff is applied on the extended model. Blank nodes cause some problems in computing the structural diff, as we have no knowledge about the relation (equal or not?) between two blank nodes from different models. In the example of Figure 2.5, it seems obvious that both models encode exactly the same semantic knowledge: there exists a thing which has a name (Max) and another one which has a phone number (123). So, it is safe to treat the blank nodes as equal across versions. But such equality is not always safe to deduce.

version A:	version B:
_:1 :hasName "Max"	_:3 :hasPhone "123"
added $a(A,B)$:	removed $r(A,B)$:
_:5 :hasPhone "123"	_:2 :hasName "Max"

Figure 2.5: **- Example B -**

Indeed, in order to identify the equality between blank nodes across models, without breaking existing semantic interpretations of the model, the concept of *Blank Node Enrichment* has been introduced. It assignes a unique identifier to each blank node by adding additional statements to the model that attach URIs as inverse functional properties to blank nodes, as illustrated in example of Figure 2.6.

Model A:
```
_:3 rdf:type foaf:Person
_:5 foaf:name "Max"
```

Model A' (blank node enriched):
```
_:3 rdf:type foaf:Person
_:3 bne:hasID rnd:1130937311/416
_:5 foaf:name "Max"
_:5 bne:hasID rnd:1130937738/417
```

Figure 2.6: **- Example C -**

Additional statements change nothing to the RDF semantic but helps to identify equal blank nodes across models.

- **Semantic Diff**: The semantic difference has to take the semantics of the used ontology language into account. The authors consider only RDF Schema as ontology language. In order to compute the

semantic RDFS diff between two ontologies, SemVersion first infers all triples from the two ontologies, computing Inf(A) and Inf(B). Then, it calculates a structural (syntactical set-based) diff on Inf(A) and Inf(B). A example of this is shown in Figure 2.7.

version A:	version B:
a rdfs:type c b rdfs:type c c rdfs:subClassOf d	a rdfs:type d b rdfs:type d c rdfs:subClassOf d e rdfs:type d
added $a(A, B)$: e rdfs:type d	removed $r(A, B)$:

Figure 4.13: Example for a Semantic Diff under RDFS semantics

Figure 2.7: **- Example D -**

Unlike the previous versioning methods for computing differences, SemDiff is actually working at a semantic level; which means that it can detect semantic differences, that are inferred from a reasoner. Unfortunately, SemVersion only works on RDF(S) and the method that they use, which is to compute the complete set of entailments of the ontology, can not be applied to OWL because this set could be infinite. Furthermore, using a reasoner is too expensive in the context of SWSEs, like Watson, where there are thousands of ontologies to consider.

2.4 Summary and Conclusions

Our research interest concerns detecting and managing semantic relations between ontologies in order to improve SWSEs' support to SWAs. Basically, we first need a formalization of these relations and then efficient methods to detect and manage them.

- **About the formalization of Semantic Relations:** As reported in section 2.1, the authors of [KA05] characterize a number of relations between ontologies giving abstract definitions and providing properties for all of them. We want to extend this work introducing a semantic structure to formalize concretely these and other kinds of relations. Furthermore, for all of them we are providing a concrete definition and efficient methods for detection.

- **About the detection of Semantic Relations:** The study of *ontology comparison*, section 2.2, and *ontology versioning*, section 2.3, have provided several approaches on how to compare two different versions of ontologies. They are, therefore, focusing on one specific relation: *versioning*.
  Some of the provided method work at syntactic and structural levels and not at semantic level. (**PROMPTDiff** [NM02], **OntoView** [KFKO02] and part of **SemVersion** [Vol06]). For this reason, recently, new methods have appeared which are working at a semantic level (**SemVersion** [Vol06] and **OWLDiff** `http://semanticweb.org/wiki/OWLDiff`). Unfortunately, they are very limited in terms of the languages they can work with. For this reason, these approaches are insufficient in scenarios like the one of SWSEs, where many ontologies of many different levels of expressivity are to be considered.
  To our best knowledge, there is no work in the literature that covers the whole topic of detecting and managing semantic relations among ontologies, that would be able to handle the variety of formats and languages that are currently present on the SW.

As a general conclusion, there is currently a need for a general framework such as KANNEL to tackle the issues related to making explicit existing, implicit semantic relations between ontologies online. The first step for such a framework is to identify and formalize the possible relations between ontologies. We realize this through the DOOR ontology of ontology relations.

Chapter 3

The DOOR-Ontology: Towards a Formalization of Ontology Relations

In this chapter, we describe our ongoing effort in describing and formalizing semantic relations that link ontologies with each other on the Semantic Web in order to create an ontology, DOOR, to represent, manipulate and reason upon these relations. DOOR is a Descriptive Ontology of Ontology Relations which intends to define relations such as inclusion, versioning, similarity and agreement using ontological primitives as well as rules. It provides a formal model for representing relations between ontologies, by defining the hierarchical structure and properties that characterize relations between ontologies. DOOR is the core of the KANNEL framework, as it provides the conceptual basis for integrating the different components and enables querying and reasoning with relations between ontologies.

Here, we provide a detailed description of the methodology used to design the DOOR ontology, as well as an overview of its content.

3.1 Methodology for Designing the DOOR Ontology

Building an ontology of relations between ontologies is a very ambitious task. It requires a deep analysis of the ontologies available online and of the literature, at different levels. Therefore, a reasonably rigorous but nonetheless flexible methodology is needed to identify, describe and formalize ontology relations and their connections, in order to build the DOOR ontology. Here, after defining some important elements that will be used in the rest of the chapter, we present the steps involved in the methodology we adopted and describe it briefly.

3.1.1 Definitions and Requirements

We consider the following definitions:

Definition 3 (Ontology) *An ontology is a set of axioms (in the sense of the description logic) over a Vocabulary VOC, where VOC is the set of the primitive terms (named entities) employed in the axioms of the ontology;*

Definition 4 (Ontology Space) *An ontology space OS, is a collection of ontologies.*

Definition 5 (Ontology Relation) *Given an ontology space OS, an Ontology Relation is any binary relation defined over OS.*

At the most general level, the design of the DOOR ontology was based on three main sources to identify relevant ontology relations:

1. We analyzed the results of SWSEs (e.g., Watson) to manually identify existing, implicit relations between ontologies in these results.
2. We considered relations described in the literature, such as the ones already mentioned in the previous sections.
3. We also included existing, explicit relations that are primitives of the OWL ontology language.

Also, ontology relations in the DOOR ontology should reflect the following important features:

- they are general enough to be applied to multiple domains;
- they are sufficiently intuitive to reflect general meaning, e.g Similar is general and reflect the fact that there is an overlap, between the two ontologies, without specifying at which levels (syntactic or semantic);
- they are formally defined to be processed automatically by inference engines;

3.1.2 Main steps of the Methodology

To design DOOR, we considered the methodology described in [GGMO01] for selecting general ontological categories and adapted it to the problem of ontology relations. As a result, we divided our approach into a number of steps, as follows:

1. Identifying the top level relations between ontologies, considering our three sources (SWSEs, literature and existing OWL primitives). At this stage, the task only consists in coming up with a list of relations that should be relevant, giving us a general overview of the different sections of the ontology. Relations such as $inclusion$, $similarity$, $incompatibility$ and $previous\ version$ are identified here.
2. Specifying the identified relations, identifying relevant variants and sub-relations. Here, our three sources of relations are also employed to derive relations at a lower level. We also use a more systematic approach, which consists in looking at ontologies (and so ontology relations) from 5 different dimensions that can characterize them:

   - **The Lexicographic level**, which concerns the comparison of the vocabularies of the ontologies.
   - **The Syntactic level**, which concerns the comparison of the sets of axioms that form the ontologies.
   - **The Structural level**, which concerns the comparison of the graph structure formed by the axioms of the ontologies.
   - **The Semantic level**, which concerns the comparison of the formal models of the ontologies, looking in particular at their logical consequences.
   - **The Temporal level**, which concerns the analysis of the evolution of ontologies in time.

   For example, considering the relation of $inclusion$ identified in the first step and that led to a property $includedIn$ in the ontology, we can specify this relation according to three different dimensions (syntactic, structural and semantic), leading to three variants of inclusion between ontologies ($syntacticallyIncludedIn$, $isHomomorphicTo$ and $semanticallyIncludedIn$) that consider the set of axioms, the graph and the formal models of the ontologies respectively. In addition, besides the systematic analysis of this relation according to the dimensions, we include in DOOR particular forms of inclusions derived from existing OWL primitives (e.g., OWL $imports$) and from the literature (e.g, $isAConservativeExtensionOf$ [GLW06]). More details about these relations are given in the next section.

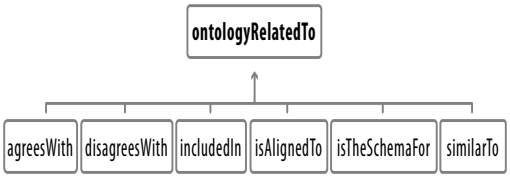

Figure 3.1: **Top Level of DOOR.**

3. Characterizing each relation by its algebraic properties. For example, the algebraic properties for similarity are that it is $reflexive$ and $symmetric$. For inclusion, we can define that it is $reflexive$ and $transitive$. Including such information in the ontology corresponds to what [GGMO01] calls defining the *ground axioms*.

4. Establishing connections between relations. The results obtained from the previous steps are mainly top-level relations with a list of variants, each of them being given algebraic properties. Here, we want to structure these lists, in particular by giving them taxonomic relations. As an example, it can be easily established that *syntacticallySimilarTo* is a sub property of *semanticallySimilarTo*. In the same way, we can indicate that a $previous\ version$ of an ontology ought to be $similar$ to it. This corresponds to defining *non-ground axioms* in [GGMO01].

5. Introducing rules to define complex relations from atomic ones. For example, the $equivalentTo$ property can be defined as *equivalentTo*(X_1, X_2)*:- includedIn*(X_1, X_2)*, includedIn*(X_2, X_1)*.*

Like in any methodology, the application of these steps should be flexible and continuous. Getting back to a previous step is sometimes necessary and, as the building of an ontology such as DOOR is a constantly ongoing effort, it should be possible to re-apply the methodology entirely to make the ontology evolve.

The intended result is an ontology made, on the one hand, of an ontologically defined and taxonomically structured set of relations, and on the other hand, of a set of rules to define complex relations. In the following we give a detailed overview of the first version of the DOOR ontology, considering only the first (ontological) part of it, as, due to its complexity, the definition of rules governing complex relations is still a work in progress and would not fit in this report.

3.2 Formal Description of DOOR

The OWL version of the DOOR ontology can be downloaded at: `http://KANNEL.kmi.open.ac.uk/ontology`.We start with describing the first level of DOOR, in Figure 3.2. The main relevant abstract relations are simply represented as sub-properties of *ontologyRelatedTo*. An ontology X is *ontologyRelatedTo* another one Y if one of the top level relations is satisfied between X and Y. The top level relations include *includedIn*, *similarTo*, *isAlignedTo*, *disagreesWith*, *agreesWith* and *isTheSchemaFor*. We clustered them in four groups and each group will be explained in more details in the next sub-sections.

3.2.1 includedIn and equivalentTo

includedIn and *equivalentTo* are two of the main ontology relations. The former represents the meaning of “an ontology contains an another one". The latter intends to convey the meaning of “two ontologies express the same knowledge". According to our methodology, these two relations have been analyzed at different levels, giving origin to different kinds of *inclusion* and *equivalence* relations. In Table 3.1, we summarize the result of these analyses:

Table 3.1: **Specialization of inclusion and equivalence relations.**

	includedIn	equivalentTo
Semantic	semanticallyIncludedIn isAConservativeExtentionOf	semanticallyEquivalentTo
Structural	isHomomorphicTo	isIsomorphicTo
Syntactic	syntacticallyIncludedIn import	syntacticallyEquivalentTo

In particular, the sub-relations of *includedIn* are defined as follows:

syntacticallyIncludedIn(X_1**,** X_2***)*** if the set of axioms of X_1 is contained in the set of axioms of X_2,which means $X_1 \subseteq X_2$.

isHomomorphicTo(X_1**,** X_2***)*** if a homomorphism exists between the RDF-graph of X_1 and the RDF-graph of the X_2.

semanticallyIncludedIn(X_1**,** X_2***)*** if the set of models of X_1 is contained in the set of models of X_2. In other words, if $X_2 \models X_1$.

isAConservativeExtentionOf(X_1**,** X_2***)*** , informally, if $syntacticallyIncludedIn(X_2, X_1)$ and all the axioms entailed by X_1 over the vocabulary of X_2 are also entailed by X_2. A more formal definition can be found in [GLW06]. The notion of conservative extension has been used in particular for ontology modularization [GHKS07].

import(X_1**,** X_2***)*** if there is an explicit statement in X_1 indicating that it imports X_2 using the $owl{:}imports$ primitive. Formally, this means that all the axioms of X_2 should be considered as contained in X_1.

The sub-relations of *equivalentTo* are defined as follows:

syntacticallyEquivalentTo(X_1**,** X_2***)*** if and only if *SyntacticallyIncludedIn(*X_1*,* X_2*)* and *SyntacticallyIncludedIn(*X_2*,* X_1*)*.

isIsomorphicTo(X_1**,** X_2***)*** if an isomorphism exists between the graph of X_1 and the graph of X_2.

semanticallyEquivalentTo if and only if *semanticallyIncludedIn(*X_1*,* X_2*)* and *semanticallyIncludedIn(*X_2*,* X_1*)*.

Finally, following our methodology, we defined the algebraic properties of each relation[1] and classified them to create a taxonomic structure relating these relations. This structure is shown in Figure 3.2[2].

3.2.2 similarTo

Ontology similarity has been described as a measure to assess how close two ontologies are [DE08]. Various ways to compute the similarity between two ontologies have been described which are relevant in different application contexts. In our work, *similarTo* is used to represent the meaning of "how an ontology overlap/cover parts of the same area of interest as another ontology". Following our methodology, *similarTo* has been analyzed and formalized at the lexicographic, structural and semantic level, giving origin to different kinds of similarity relations (see Table 3.2).

[1] Since these are fairly obvious, we do not detail them.

[2] The arrows represent the subPropertyOf relation. For example, syntacticallyEquivalentTo is a sub property of semanticallyIncludedIn.

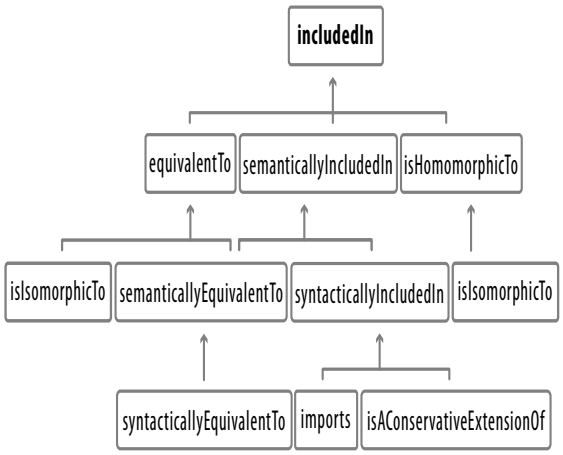

Figure 3.2: **Taxonomy for includedIn and equivalentTo**

Table 3.2: **Specialization of the similarity relation**

	SimilarTo
Semantic	semanticallySimilarTo MappingSimilarTo
Syntactic	syntatticallySimilarTo
Lexicographic	LexicographicSimilarTo

To define these relations, we need to introduce the following elements: given two ontologies X_1 and X_2, we denote by *LC(X_1, X_2)* the set of axioms of X_1 that are logical consequences of X_2 and by *Voc(X_1)* the vocabulary of X_1. The following definitions depend on a threshold $T > 0$.

semanticallySimilarTo(X_1, X_2), if

$$\frac{|LC(X_1, X_2) \bigcap LC(X_2, X_1)|}{max(|X_1|, |X_2|)} \geq T$$

syntacticallySimilarTo(X_1, X_2), if

$$\frac{|X_1 \bigcap X_2|}{max(|X_1|, |X_2|)} \geq T$$

lexicographicallySimilarTo(X_1, X_2), if

$$\frac{|Voc(X_1) \bigcap Voc(X_2)|}{max(|Voc(X_1)|, |Voc(X_2)|)} \geq T$$

Finally, in addition to the relations defined above, we also consider a similarity relation that relies on the existence of an alignment between the two ontologies. Indeed, **mappingSimilarTo** is a relation that links two ontologies X_1 and X_2 if there exists an alignment from X_1 to X_2 and this alignment covers a substantial part of the vocabulary of X_1 (i.e., a proportion greater than a threshold T). Note that, since alignments can be unidirectional, $mappingSimilarTo$ differs from the other similarity functions by not being symmetric.

Finally, we have classified the relations in Table 3.1 to create the taxonomic structure shown in Figure 3.3. In the Figure 3.3, the dashed elements represent relations regarding to the Versioning of the ontology.

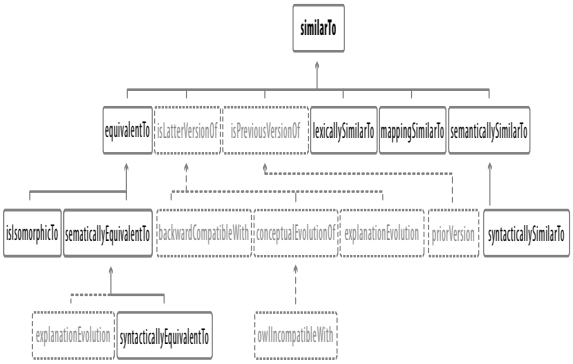

Figure 3.3: **Taxonomy for similarTo.**

3.2.3 Versioning

Versioning is a temporal relation that concerns the evolution of an ontology. In [KF01], the ontology versioning problem has been defined as *the ability to handle changes in ontologies by creating and managing different variants of it.*

An ontology can evolve over time in different directions, e.g. *lexicographic*, changing the names of some resources, *syntactic*, adding or removing axioms, *semantic*, changing the definition of some concepts or simply adding or removing axioms. Therefore, the new ontology could be equivalent or totally different from the previous one. When we analyze different ways of linking two ontologies by the versioning relation, the two following sentences are suggested immediately: "X_1 is the previous version of the X_2" or "X_2 is the latter version of the X_1". These two typical pieces of knowledge are represented in the DOOR ontology by the relations *isPreviousVersionOf* and its inverse *isLatterVersionOf* respectively.

Conforming to our methodology, the *isPreviousVersionOf* and *isLatterVersionOf* relations have been analyzed and formalized, to identify sub-relations and variants. In Table 3.3 we summarize the result of this analysis.

Table 3.3: **Specialization of the versioning relations.**

	isLatterVersionOf	isPreviousVersionOf
Temporal	conceptualEvolutionOf explanationEvolutionOf backwardCompatibleWith owl:IncompatibleWith	priorVersion
Semantic	conceptualEvolutionOf	
Syntactic	explanationEvolutionOf	

According to [KFKO02, HP04, Hef01] the modification of an ontology can lead to two different types of evolutions: being a conceptual change, meaning that the model of the ontology changed, or being an explanation change, meaning that the modifications happened only at a syntactic level, without affecting the model of the ontology. Therefore, we specialized the *isLatterVersionOf* relation into

conceptualEvolutionOf(X_1, X_2***)*** if X_1 is a latter version that is not semantically equivalent to X_2.

explanationEvolutionOf(X_1, X_2***)*** if X_1 is a latter version that is semantically equivalent to X_2

These two relations will lead to the definition of rules to infer them from equivalence and other versioning relations.

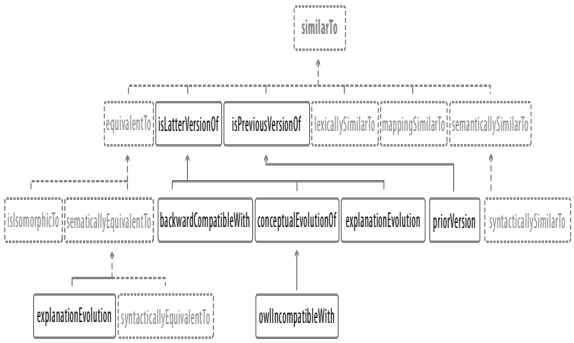

Figure 3.4: **Taxonomy for versioning relations.**

In addition, the OWL ontology properties $priorVersion$, $backwardCompatibleWith$ and $incompatibleWith$ represent explicit relations between versions of ontologies and are included in DOOR as sub-properties of $isLatterVersionOf$ and $isPreviousVersionOf$.

To complete this section of the DOOR ontology, we can classified the relations in Table 3.3 as shown in Figure 3.4.

Indeed, according to [KFKO02, HP04, Hef01] the modification of an ontology can lead to a new version which is completely different from the original one. But in practice, by analyzing Watson's ontology repository, it is almost always possible to establish a similarity between the two ontologies, at least at the lexicographic level. Due to this fact, we chose to consider the versioning relations to be sub-properties of $similarTo$, to indicate that two different versions of the same ontology should, to some extent, be similar. Moreover, in accordance with its definition, the *explanationEvolutionOf* relation is a sub-property of *semantticallyEquivalentTo*.

3.2.4 Agree and Disagree

Based on the formal measures of the agreement and disagreement between ontologies defined in [d'A09], we introduce the *agreesWith* and *disagreesWith* relations in DOOR. Informally, the former holds the general meaning of "to have the same opinion about something". In other words, it connects two ontologies, sharing the same knowledge partially and is therefore very related to the *similarTo* and the *equivalentTo* relations. The latter indicates that the ontologies "contradict each other" to a certain extent, these contradictions appearing at various levels. Envisaged sub-relations for these two relations are listed in Table 3.4.

Table 3.4: **Specialization of agreesWith and disagreesWith.**

	agreeWith	disagreeWith
Temporal	backwardCompatibleWith	owlIncompatibleWith
Semantic	sematicallyEquivalentTo sematicallySimilarTo	hasDisparateModeling incompatibleWith incoherentWith incosistentWith
Syntactic	syntacticallyEquivalentTo syntacticallySimilarTo explanationEvolution	

In this Table, all the sub-relations of $agreesWith$ have already been defined before. We add a few relations to express specific ways for ontologies to disagree, all related to the semantic dimension of the ontologies.

incompatibleWith(X_1***,*** X_2***)*** if *incoherentWith(*X_1*,* X_2*)* or *inconsistentWith(*X_1*,* X_2*)*.

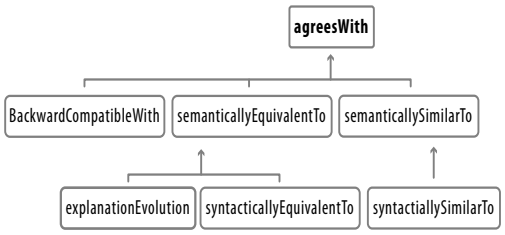

Figure 3.5: **Taxonomy for the agreement relations.**

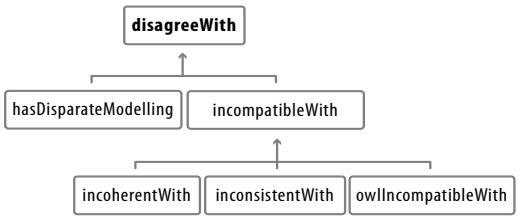

Figure 3.6: **Taxonomy for the disagreement relations.**

incoherentWith According to [QH07] an ontology X_1 is incoherent if and only if there is an unsatisfiable concept name in X_1. Therefore, two ontologies are *incoherent* with each other if, when they are merged, they generate an incoherent result.

inconsistentWith According to [BQL07] an ontology X_1 is inconsistent if it has no model. Therefore, two ontologies are *inconsistent* with each other if, when they are merged, they generate an inconsistent result.

hasDisparateModeling Two ontologies are considered to have disparate modeling if they represent corresponding entities in different ways, e.g. as an instance in one case and a class in the other. For example, Lion as instance of the class Animal and Lion as subclass of Animal.

owl:IncompatibleWith It comes from OWL language [PSHH04].

Finally, we have also classified the relations in Table 3.4 as shown in Figures 3.5 and 3.6.

3.2.5 Other Relations

Analyzing Watson's ontology repository we found out that there are many documents which only represent the TBox of an ontology and others representing just the ABox. This is captured through the *isTheSchemaFor* relation. *isAlignedTo* relation links ontologies for which exists an alignment.

3.3 Conclusions

In this chapter, general relationships between ontologies have been examined. In particular, we have chosen to consider well-known relations in the literature, as well as the ones needed to support the development of Semantic Web Applications. To achieve that, we adapted an ontology building methodology for the construction of DOOR, an ontology of relations between ontologies. The first version of the DOOR ontology is available in OWL at `http://KANNEL.kmi.open.ac.uk/ontology`. This ontology describes relations both from the point of view of their taxonomic structure and from the point of view of their formal

definitions, providing the formal properties to describe them as well as a set of rules to derive complex relations from other relations.

In the next chapter we describe KANNEL, a framework for detecting and managing ontology relationships for large ontology repositories. The DOOR ontology plays a fundamental role in KANNEL, not only to provide an explicit representation on ontology relations, but also to supply meta-information that offers several advantages, among which the possibility to reason upon ontologies and their relations. This possibility provides a relevant support for the development of Semantic Web Applications, which can use the Semantic Web as a large-scale knowledge source [dMea08].

The development of DOOR is obviously a continuous task, which requires a proper assessment of each version. For this reason, we plan to test and validate the first version presented here, in particular by populating it with automatically detected relations between ontologies in Watson.

Chapter 4

The KANNEL Framework

In this chapter we provide an overview of the architecture of KANNEL, an ontology based framework for making explicit implicit relations between ontologies in large ontologies repositories such as Watson, presenting the different parts of it and the role of DOOR in it.

4.1 KANNEL: Architecture

KANNEL [All09] is a framework for detecting and managing ontology relations for large ontology repositories, such as Watson[1].

It is an ontology-based system where the DOOR ontology plays an important role, providing an explicit representation of the implicit relations between ontologies. We have designed an architecture for this framework, as depicted in Figure 4.1. As shown in Figure 4.1, the DOOR Ontology separates the on-line part of the architecture–providing APIs and services that relies on a reasoner–from the off-line part–detecting relations in the repository and populating the ontology. In particular, the REASONING component provides a suite of services to support users in exploring and making use of ontology relations on the Semantic Web. This component allows the system to be queried for relations, which have been either detected by the Relation Discovery component, or inferred through ontological and/or rule-based reasoning over the DOOR ontology. The offline part is based on three components: the *Control Component (CC)*, the *Detecting Component (DC)* and the *Populating Component (PC)*. As a first step, the CC selects from the Ontology Repository ontologies that need to be evaluated to establish potential relations. Then, the selected sets of ontologies are processed by the DC, which contains the main mechanisms to discover the possible relations between ontologies, relying on the definitions provided in this report. Finally, the PC populates the semantic structure with the detected relations. What is obtained is a set of automatically discovered relations, represented as part of the DOOR ontology so that the reasoner used in the system can infer new relations from the ontological and rule based knowledge included in the ontology. As such, DOOR provides meta-information on the ontology repository in KANNEL.

In the following sections, we present various detection mechanisms that have been implemented to extract from a collection of onologies, specific semantic relations to populate the DOOR ontology.

4.2 Inclusion and Equivalent

Inclusion is an example of atomic or simple relation in DOOR. It links together two ontologies if one is *contained* in the other. We consider here two of the identified inclusion relations: syntactic (considering the set of axioms asserted in the ontologies) and semantic (also considering the entailments of the ontologies). Specifically, given two ontologies X_1 and X_2, we formally define:

[1] The management part is not currently dealt with.

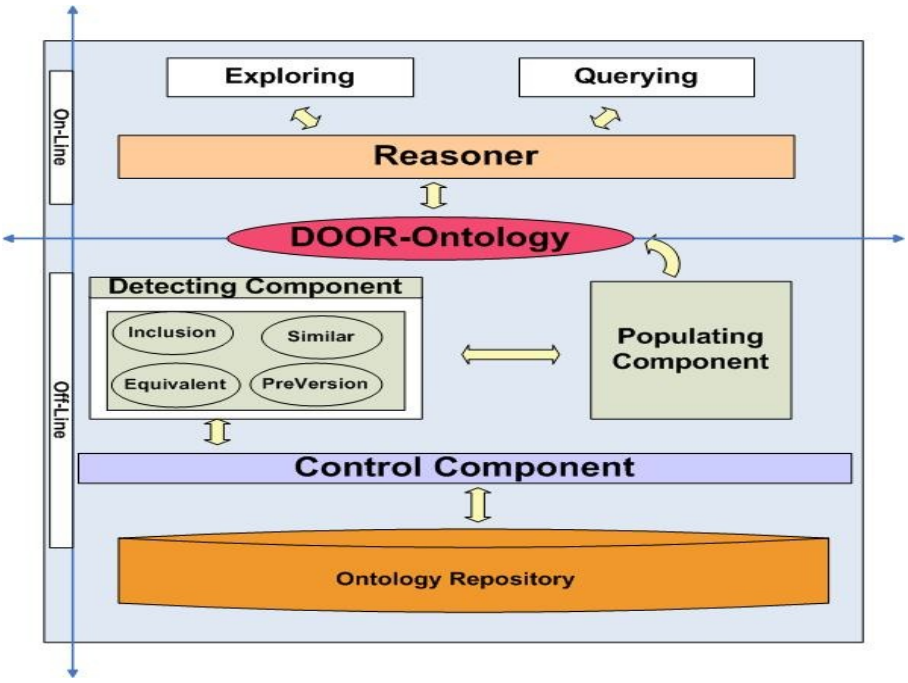

Figure 4.1: **Architecture of the KANNEL framework.**

syntacticallyIncludedIn(X_1***,*** X_2***)*** if the set of axioms of X_1 is contained in the set of axioms of X_2,which means $X_1 \subseteq X_2$.

semanticallyIncludedIn(X_1***,*** X_2***)*** if the set of models of X_1 is contained in the set of models of X_2. In other words, if $X_2 \models X_1$.

Equivalence is a complex relation which can be defined through a rule of the general form $Equivalence(X_1, X_2) :- Inclusion(X_1, X_2), Inclusion(X_2, X_1)$. In particular, for syntactically equivalent we use *syntacticallyIncludedIn* and for semantically equivalent we use *semanticallyIncludedIn*.
Detecting equivalence (syntactic and semantic) is particularly important when ranking ontologies: the same ontology should not been ranked at several different places in the result set [HA06].

4.2.1 Initial Evaluation

To provide an initial evaluation of the current implementation of the above detection mechanisms, typing the query "Person", we collected 20 pairs of ontologies from Watson's repository and executed the above detection methods on them. In Table 1 the results for each evaluated method are shown.

Method	Nb. Relations	Average Time (ms)	Manual eval.
SyntacticInclusion	8	2	8
SemanticInclusion	9	80	9
SyntacticEquivalent	4	5	4
SemanticEquivalent	6	180	6

4.3 Similar Ontologies

Similarity is a difficult relation to detect, first because it does not have a formal definition, but more importantly because different similarity measures can be considered, depending on the task at hand. For this reason, we present an initial collection of similarity relations, each with their own detection mechanism. We first report the main assumptions on which our measures of similarity depend.

- **A1:** The ontologies vary by changing some definitions or adding new axioms or deleting axioms [KF01]. However, these different ontologies might not have the same namespace, making their axioms incomparable. Our assumption is therefore that we can strip out the first part of the resources' names, which corresponds to the namespace of the resource. By doing this, we only compare the ontologies within the namespaces. For example using "namespace1" for $http : //www.vistology.com/ont/tests/student1.owl$ and "namespace2" for $http : //www.vistology.com/ont/tests/student2.owl$, the axioms

  - X_1 : namespace1#PhdStudent SubclassOf namespace1#MemberStaff
  - X_2 : namespace2#PhdStudent SubclassOf namespace2#MemberStaff

  are compared as namespace1 and namespace2 were equal. In this case, they are exactly the same axioms.

  We do not provide a formal proof of the validity of this assumption. We give an informal argumentation considering the hypothesis does not convey any additional meaning that would impact the comparison of ontologies for similarity. Indeed, looking at the following example:

  - X_1 :namespace1#PhdStudent **SubclassOf** namespace1#MemberStaff
    namespace1#MemberStaff **SubclassOf** namespace1# $\forall role.Academic$
  - X_2 :

    namespace2#PhdStudent **SubclassOf** namespace2#MemberStaff
    namespace2#MemberStaff **SubclassOf** namespace2# $\forall role.Academic$

  It appears clearly that X_1 and X_2 should be considered the same ontology (they describe the same intended model) while in this other example:

  - X_1 :

    namespace1#PhdStudent **SubclassOf** namespace1#MemberStaff
    namespace1#MemberStaff **SubclassOf** namespace1# $\forall role.Academic$

  - X_2 :

    namespace2#PhdStudent **SubclassOf** namespace2#MemberStaff
    namespace2#MemberStaff **SubclassOf** namespace2# Person

  the ontologies are different, but very similar. Such a conclusion could not be drawn if we were taking into account the namespaces.

- **A2:** We also consider a syntactic normalization of the terms of the ontologies' vocabularies. For example, "MemberStaff", "Member-Staff", "Member_Staff" or "memberstaff" are all considered the same.

- **A3:** Our measures of similarity are applied over the ontologies which share (part of) their vocabularies. By doing this, we avoid comparing ontologies that formalizes different domains.

4.3.1 Measuring Similarity Between Ontologies

Informally, *similarity* is a measure that expresses how close two ontologies are. [DE08] describes various ways to compute the similarity between two ontologies, relevant in various application contexts, but not yet evaluated in the context of SWSEs. [AdM09] describes the DOOR ontology in the context of SWSEs. DOOR considers three types of binary relations to express different level of similarity between two ontologies: 1) *lexicographicallySimilarTo*; 2) *syntacticallySimilarTo*; and 3) *semanticallySimilarTo*. We also add here an additional similarity relation: *KeyConceptOntologySimilarTo*.

Here, we define four measures of similarity on which methods for automatically detecting the above relations are based on. To do this, we need to introduce the following elements: given two ontologies X_1 and X_2, we denote by *LC(X_1, X_2)* the set of axioms of X_1 that are inferred by X_2, and by *Voc(X_1)* the vocabulary of X_1. To decide whether two ontologies share a similarity *relation*, we apply a given threshold to each of them. Moreover, *reflexive* and *symmetric* are the main algebraic properties that characterize any type of similarity described below.

4.3.2 Lexicographic Similarity

Let's consider the ontologies X_1: C⊑B, B⊑A, B⊑ $\exists R.D_1 \sqcup \exists R.D_2$ and X_2: C⊑B, B⊑A,C⊑A, B⊑ $\exists R.(D_1 \sqcup D_2), D \sqsubseteq E$. The vocabulary of X_1 is C, B, A, R, D_1, D_2 while the vocabulary of X_2 is $C, B, A, R, D_1, D_2, D, E$. From the example, it can be seen that the two ontologies share most of their vocabularies. Based on this idea, we formally define the *Lexicographic Similarity* measure as follows:

lexicographicSimilarity(X_1, X_2) =

$$\frac{|Voc(X_1) \bigcap Voc(X_2)|}{max(|Voc(X_1)|, |Voc(X_2)|)}$$

4.3.3 Syntactic Similarity

Let's consider the ontologies X_1: C⊑B, B⊑A, B⊑ $\exists R.D_1 \sqcup \exists R.D_2$ and X_2: C⊑B, B⊑A, B⊑ $\exists R.D_1 \sqcup \exists R.D_2, D \sqsubseteq E$. From the example, it can be seen that the two ontologies share most of their axioms. Based on this idea, we formally define the *Syntactic Similarity* measure as follows:

syntacticSimilarity(X_1, X_2) =

$$\frac{|X_1 \bigcap X_2|}{max(|X_1|, |X_2|)}$$

4.3.4 Semantic Similarity

Let's consider the ontologies X_1: C⊑B, B⊑A, B⊑ $\exists R.D_1 \sqcup \exists R.D_2$ and X_2: C⊑B, B⊑A,C⊑A, B⊑ $\exists R.(D_1 \sqcup D_2), D \sqsubseteq E$. From the example, while the ontologies do not share all of their axioms, the ones not in common can be inferred from each ontology.

Based on this idea, we formally define the *Semantic Similarity* measure as follows:

semanticSimilarity(X_1, X_2) =

$$\frac{|LC(X_1, X_2) \bigcap LC(X_2, X_1)|}{max(|X_1|, |X_2|)}$$

Please note: when comparing two ontologies, to avoid computing all consequences, which is not always possible [HS05], our methods only check if the set of axioms of the first ontology is entailed by the second

ontology and vice-versa.

4.3.5 KeyConcepts Similarity

Informally, KeyConcepts of ontologies are n concepts which can be considered as Ôbest descriptorsÕ of the ontology. According to the definition of key concepts given in [SPd08], we define two measures of similarity: 1) *unrankedKeyConceptOntologySimilar* ; 2) *rankedKeyConceptOntologySimilar*, both described below. In the following, we use:

- KC(X_j, n) to indicate the first *n* key concepts of the ontology X_j;
- A_{X_j} to indicate a generic element belonging to KC(X_j, n);
- rank(KC(X_j, n), i) to indicate a function which return the degree of importance of the key concept *i* in KC(X_j, n).

unrankedKeyConceptOntologySimilar considers the key concepts of an ontology without their degree of importance. So, KC(X_j, n) is just a set of elements. We formally define this measure as follows:

unrankedKeyConceptOntologySimilar(X_1***,*** X_2***)*** =

$$\frac{|KC(X_1,n) \bigcap KC(X_2,n)|}{n} \quad (1)$$

The equation (1) can be seen as a particular case of *lexicographicSimilarity*, that is restricted to the key concepts instead of whole vocabularies. This could be an advantage in terms of time to calculate it. Also, by limiting the comparison to the key concepts, this measure also indirectly takes into account the content and structure of the ontologies, in addition to their vocabularies.

rankedKeyConceptOntologySimilar considers the key concepts of an ontology with their degree of importance. In this case, KC(X_j, n) is a list of elements. The order expressed by the list indicate the order of importance of the concepts with respect to the considered ontology. For example, given two ontologies X_1 and X_2 with KC(X_1, 3)= [$A_{X_1}, B_{X_1}, C_{X_1}$] and KC($X_2$, 3)= [$C_{X_2}, B_{X_2}, A_{X_2}$] respectively. Assigning the highest priority to the first element on the left, then, it can be seen that A_{X_1} represents the most important concept in X_1 while the same concept in X_2, A_{X_2}, represents a less important concept in X_2. Based on this aspect of importance, we define the *rankedKeyConceptSimilar* measure according to the following elements: 1) its value is 1 if we have the same list and the same order; 2) its value is 0 if we have different lists, that is the intersection of the two list is empty; 3) its value is greater than zero or smaller than 1 if it could be one of the two cases: (a) KC(X_1, n) = KC(X_2, n) but different order; or (b) KC(X_1, n) $\cap$ KC(X_2, n) $\neq \emptyset$ with the order of KC(X_1, n) could be different from the KC(X_2, n) one. Formally, we have:

rankedKeyConceptOntologySimilar(X_1***,*** X_2***, n)*** =

$$\frac{\sum_{i=1}^{n} keyConceptSimilarity(KC(X_1,n), KC(X_2,n), i)}{n}$$

where *keyConceptSimilarity(KC(*X_1*, n), KC(*X_2*, n),i)* calculate the distance between the same key concept, *i*, in two different key concepts lists. Formally, we define it as follows:

$$keyConceptSimilarity(KC(X_1,n), KC(X_2,n), i) = \begin{cases} 1 - \frac{|rank(KC(X_1,n),i) - rank(KC(X_2,n),i)|}{n-1} \\ if\ i \in KC(X_1,n) \cap KC(X_2,n) \\ \\ 0\ otherwise. \end{cases}$$

An example about rankedKeyConceptOntologySimilar ontologies. Let's X_1 and X_2 be two ontologies, with KC(X_1, 3)= [$Person_{X_1}, Organization_{X_1}, Publication_{X_1}$] and KC($X_2$, 3)= [$Publication_{X_2}, Organization_{X_2}, Person_{X_2}$] respectively. Let's assign the degree of importance as follows:

1. rank(KC(X_1, 3), $Person_{X_1}$) = 3;
2. rank(KC(X_1, 3), $Organization_{X_1}$) = 2;
3. rank(KC(X_1, 3), $Publication_{X_1}$) = 1;
4. rank(KC(X_2, 3), $Publication_{X_2}$) = 3;
5. rank(KC(X_2, 3), $Organization_{X_2}$) = 2;
6. rank(KC(X_2, 3), $Person_{X_2}$) = 1;

then

1. $keyConceptSimilarity(KC(X_1,3)KC(X_2,3), Person) = 1 - \frac{|3-1|}{2} = 0$
2. $keyConceptSimilarity(KC(X_1,3)KC(X_2,3), Organization) = 1 - \frac{|2-2|}{2} = 1$
3. $keyConceptSimilarity(KC(X_1,3)KC(X_2,3), Publication) = 1 - \frac{|1-3|}{2} = 0$

Therefore, ***rankedKeyConceptOntologySimilar(***X_1***,*** X_2***, 3)*** = 0.33. It means that, although the ontologies share the same key concepts they have different meaning, that is, they reflect two different formalizations. In fact, if we look at just vocabulary, they contain the same set of resources. If we look at key concept level, Publication is the most important concept for X_2 while Person is for X_1. So, X_2 could be, for example, an ontologies about the publications that are published by the people while X_1 could be, instead, an ontology about the people that publish. Therefore, they are conceptually dissimilar.

While there is a clear distinction, in terms of differences and advantages, among the first three measures, it could be not so immediate between the first three and the key concepts based ones.
There are a number of advantages related to *Key Concept Similarity* measures with respect to the others presented in this document. In particular, in contrast to the *SemanticSimilarity* measure, which involves an OWL reasoner with its very high computation complexity, even for small and not complex ontologies, the key concepts measures are both very simple to compute and with computation complexity linear with respect to the number the key concepts. Compared to *SyntacticSimilarity* and *LexicographicSimilarity* which have a polynomial complexity with respect to the cardinality of the set of axioms and of the vocabulary, respectively, the key concepts measures are linear in number of the key concepts, which are a subset of the vocabulary of the ontology. In addition, while comparing essentially at the level of the vocabulary, the restriction to key concepts also indirectly takes into account the content and structure of the ontologies, therefore not limiting the similarity to the lexical aspect only.

4.4 Ontology Versioning

There exist a number of large repositories and search engines collecting ontologies from the web or directly from users. While mechanisms exist to help the authors of these ontologies manage their evolution locally, the links between different versions of the same ontology are often lost when the ontologies are collected by such systems. By inspecting a large collection of ontologies as part of the Watson search engine, we can see that this information is often encoded in the identifier of the ontologies, their URIs, using a variety of conventions and formats. We therefore devise an algorithm, the Ontology Version Detector, which implements a set of rules analyzing and comparing URIs of ontologies to discover versioning relations between ontologies. Through an experiment realized with 7000 ontologies, we show that such a simple and extensible approach

actually provides large amounts of useful and relevant results. Indeed, the information derived from this algorithm helps us in understanding how version information is encoded in URIs and how ontologies evolve on the Web, ultimately supporting users in better exploiting the content of large ontology repositories.

A number of studies have intended to tackle some of the challenges raised by ontology versioning, from both theoretical and practical points of views. At the theoretical level, studies have targeted ontology versioning in order to provide a theoretically semantic model for managing ontologies in distributed environments, such as the Web [KF01]. According to [KF01], the ontology versioning problem has been defined as *the ability to handle changes in ontologies by creating and managing their own variants/mutants/versions.* In other words, ontology versioning means that there are multiple variants of an ontology around and that these variants should be managed and monitored. Accordingly, tools such as *Evolva* [**?**] have been developed to support the developers of ontologies in making them evolve and in managing the versions locally. However, such systems use different ways to represent and codify version information, which is not transfered when the ontologies are collected and made accessible through online repositories. Standards such as OWL and OMV [HPea05] include primitives to encode version information as ontology annotations. However, such standards are not universally used and ontology developers rarely make the effort of applying such standards. Instead, they tend to codify information related to the version of an ontology directly in its URI. Indeed, typing the query "*metadata*" currently gives 1356 results in the Watson search engine[2] (valid on the 20/08/2009). However, only inspecting the URIs in the first page of results, we can see that many of these documents (e.g., `http://loki.cae.drexel.edu/~wbs/ontology/2004/01/iso-metadata` and `http://loki.cae.drexel.edu/~wbs/ontology/2003/10/iso-metadata`), represent different versions of the same ontology.

In this section, we present an algorithm, the *Ontology Version Detector* (OVD) which tries and detect different ways (i.e., different conventions) for encoding version information in ontology URIs in order to derive versioning links between ontologies within a large repository. It relies on a comparison of the URIs of ontologies to detect number differences, which can represent version numbers (e.g., v1.2, v3.6), dates (e.g., 2005/04, 01-12-1999) or other types of versioning information (e.g., time-stamps). One of the advantages of such an approach is that it is based on a set of rules, each encoding a particular pattern for the representation of version information and so, if missing patterns are observed in the collection, they can easily be added and taken into account by the algorithm. In this report, we detail the set of rules derived from our observations using the Watson ontology search engine.

We conducted an experiment applying OVD on a sub-set of the Watson repository of ontologies containing about 7000 ontologies. While we are aware that the approach implemented by OVD has a number of limitations (i.e., it only looks at the information encoded in the URI through numbers), this experiment shown that a large amount of versioning links implicitly encoded in the URIs of the ontologies can be correctly detected. Indeed, this experiment resulted in 155,589 versioning links, representing 1,365 "evolving ontologies" and which have been evaluated with an estimated precision of 51.2%. In addition, the analysis of these results allows us to identify ways to overcome the limitations of OVD, to better understand how version information is encoded in URIs and to assess how ontologies evolve on the Web, ultimately providing valuable information for the users of ontology repositories.

In the next section, we describe a number of examples and general patterns we observed in the Watson collection of ontologies. Section 3 then details our OVD algorithm for detecting and comparing such patterns. Our experiment on applying OVD is presented in Section 4. Finally, Section 5 discuss the conclusions and future work.

4.4.1 Identifying Version Information Patterns

Analyzing a representative sample (nearly 1000) of ontologies from Watson's ontology repository, we have, manually, identified many ontology URIs containing information concerning the version of the ontology. In this report, we focus on particular versioning patterns. Specifically, we only discuss three classes of URIs that

[2] http://Watson.kmi.open.ac.uk

can be compared to establish versioning links between URIs: *Class A*, where the versioning information is encoded in a single number; *Class B*, where the versioning information is expressed by two numbers, which are often the month and year of a date; and *Class C* where the versioning information is expressed by three numbers, which always correspond to complete date.

Class A: version information expressed by one number

These are simplest and most frequent cases: when the comparison of two URIs would only show a difference in one number. In many examples, this number represents a very simple version number, like in the following example:

Example 1:

1. `http://www.vistology.com/ont/tests/student1.owl`;
2. `http://www.vistology.com/ont/tests/student2.owl`;

However, there can be many variants of such a pattern. In the following example, a time-stamp is used to mark a particular version of the ontology:

Example 2:

`http://160.45.117.10/semweb/webrdf/#generate_timestamp_1176978024.owl`

`http://160.45.117.10/semweb/webrdf/#generate_timestamp_1178119183.owl`

Class B: version information expressed by two numbers

Under this category, we find more classical ways to represent version numbers (with a number of the major revision and a number of the minor revision), like in the following example:

Example 3:

1. `http://lsdis.cs.uga.edu/projects/semdis/sweto/testbed_v1_1.owl`
2. `http://lsdis.cs.uga.edu/projects/semdis/sweto/testbed_v1_4.owl`

However, a majority of the URIs using two numbers to represent version information use a date format that includes the year and the month. In the following example, the year is the first element to be encoded.

Example 4:

`http://loki.cae.drexel.edu/~wbs/ontology/2003/02/iso-metadata`

`http://loki.cae.drexel.edu/~wbs/ontology/2003/10/iso-metadata`

`http://loki.cae.drexel.edu/~wbs/ontology/2004/01/iso-metadata`

`http://loki.cae.drexel.edu/~wbs/ontology/2004/04/iso-metadata`

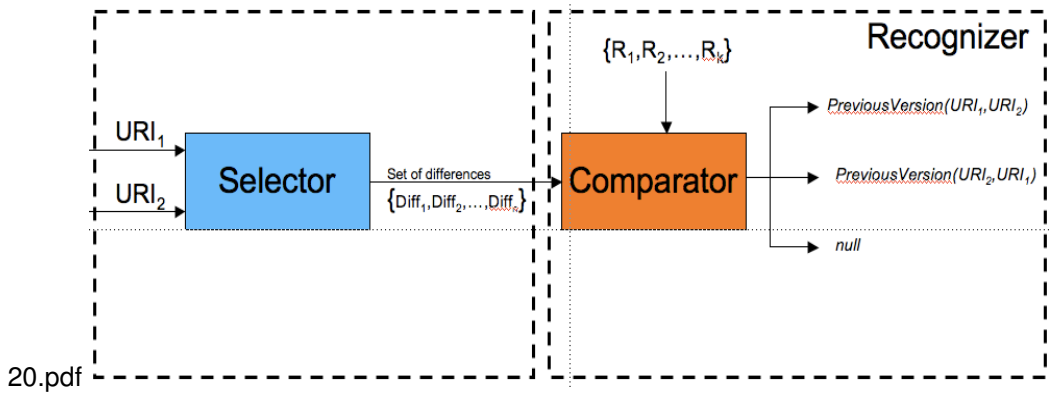

Figure 4.2: **The main steps of OVD.**

However, there can be four different ways to combine the month and the year: 1) Big endian form (yyyy/mm), in which the year is expressed by four digits; 2) Little endian form (mm/yyyy), in which the year is expressed by four digits; 3) Big endian form yy/mm, in which the year is expressed by only two digits; 4) Big endian form (mm/yy), in which the year is expressed by only two digits. Since we did not encounter examples where the year was expressed with two digits only, we ignore this case in this report, but rules to cover such patterns can easily be added to the OVD algorithm.

Class C: version information expressed by three numbers

Under this category we only found cases in which the versioning information is represented through a date format structure using the day, the month and the year. The example below is based on the big endian form (yyy-mm-dd), but as for the cases above, little endian forms (and even middle endian forms, mm-dd-yyy) could be employed.

Example 5

`http://ontobroker.semanticweb.org/ontologies/ka2-onto-2000-11-07.daml`

`http://ontobroker.semanticweb.org/ontologies/ka2-onto-2001-03-04.daml`

4.4.2 The Ontology Version Detector Algorithm

As shown in the previous section, there is no official and implemented standard or canonical form to represent version information through the URIs. However, the ontology engineers use "semi-rational" criteria to rename new versions, leading to the emergence of common patterns. The goal of the OVD algorithm is to provide a general mechanism to detect such patterns.

Overview

A general oververview of OVD is described in Figure 4.2. There are two main steps to the process: the *Selector* component compares URIs to extract sets of numerical differences between them, and the *Recognizer* component detects known patterns in these differences, encoded as a set of rules to generate versioning relations between ontologies. In this report, we focus on particular versioning patterns. In particular, the ones in which the versioning information is codified by single number or date with two-digit and three-digit format. The Selector components takes as input a set $\{URI_1, URI_2, ..., URI_n\}$ of URIs of ontologies and returns of set of numerical differences between pairs of URIs. More precisely, to a pair of URIs (URI_i, URI_j), the

Selector associate an ordered list of numerical differences represented as pairs (n_{ik}, n_{jk}) where n_{ik} is a number part of URI_i and n_{jk} is a number which replaces n_ik in URI_j. Pairs of URIs for which differences appear in other parts than numbers are simply discarded.
To realize this task, the Selector first *sequences* each URIs in a chain of sections, separating parts that represents numbers from the ones that represent other elements. If two URIs contain the same number of number sections and non-number sections, and if all the non-number sections are equal, the numerical differences are straightforwardly extracted from comparing the number sections of the two URIs with each other. For example, the URIs:

- `http://loki.cae.drexel.edu/~wbs/ontology/2003/10/iso-metadata`
- `http://loki.cae.drexel.edu/~wbs/ontology/2004/01/iso-metadata`

are sequenced in the following way:

- `http://loki.cae.drexel.edu/~wbs/ontology/` ‖ `2003` ‖ `/` ‖ `10` ‖ `/iso-metadata`
- `http://loki.cae.drexel.edu/~wbs/ontology/` ‖ `2004` ‖ `/` ‖ `01` ‖ `/iso-metadata`

Since all the non-number sections are equal and all the number sections are different, the generated differences correspond to the list of pairs of number sections: $[(2003, 2004); (10, 01)]$. It is important to notice here that the number of digits used to represent each number needs to be kept in the representation. Indeed, this information is used as part of the pattern recognition and, for example, '1' should not be considered equivalent to ''01'. We use the notation $|n|$ to indicate the number of digits used in the string representation of the number n in a set of differences.

The second step of the process, the Recognizer component, takes as input the list of differences between pairs of URIs and derive from them versioning relations that should hold between the corresponding ontologies. Versioning relations are represented here as (ordered) pairs of URIs, with $[URI_i, URI_j]$ meaning that URI_j is a more recent version of URI_i. To realize this task, the Recognizer rely on a set of rules that detect specific patterns in the differences, such as the ones identified in Section 2. Below, we detail the set of rules we have defined from our anlysis of the Watson repository.

Versioning Relation Detection Rules

The *Recognizer* implements a set of rules which are designed to cover the different way that are used to rename a new version, that is, the version information patterns. The rules presented here reflect the main characteristics of the classes of URIs discussed in Section 4.4.1, but can be easily extended for additional patterns (many of them being easily derived from the existing rules) but are ignored here as they do not appear in the considered collection of ontologies (e.g., cases where the year is represented with 2 digits, where the date is represented in middle endian form, or the version number is represented with 3 components–x.y.z). Each rule considers a pair of URIs (URI_i, URI_j) and the corresponding list of differences $D_{ij} = [(n_{i1}, n_{j1}), ...]$ to derive a probable versioning relation between URI_i and URI_j.

Class A

As indicated before, Class A corresponds to the most straightforward case: there is only one numerical difference between two URIs (i.e. the cardinality $|D_{ij}|$ of D_{ij} is 1). In this case, we only need to compare the number in question to derive which version came first.

Based on the above concrete examples, a first class of cases can be detected considering the circumstance where there is only one difference, that is, Difference(URI_1, URI_2))==1 and it is numerical. To this end, we formalize the following rule R1:only one difference that is, Difference(URI_1, URI_2))==1 and it is numerical. So, we set the rule which take such difference into account. It can be formalized as in the following:

R1 IF ($|D_{ij}| = 1$) THEN
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

The rule R1 addresses the cases such as the ones shown in Example 1 and Example 2.

Class B

Class B is a more complicated example, as it corresponds to the cases where two numbers differ from one URI to the other. Therefore, to realize an appropriate comparison, it is first needed to find which number is the most significant. We distinguish two main cases: 1- the version information corresponds to a version number, in which case the number on the left is more significant, or 2- the version information corresponds to a date including the year and month, in which case, the year is more significant.

Therefore, in order to distinguish these different situations, we need to be able to recognize and year and a month from any other number. We define the following 2 predicates, $year(n)$ and $month(n)$, with return true if the number n can be a year or a month respectively:

$$year(n) :: |n| = 4 \text{ and } 1995 \leq n \leq current\_year$$
$$month(n) :: |n| = 2 \text{ and } 01 \leq n \leq 12$$

These conditions assume that ontologies can only have been created from 1995 to the present year, that months are always represented with 2 digits and years with 4 digits. While these assumptions might appear restrictive, they reflect our observations on the Watson collection of ontologies and help in avoiding unnecessary noise.

Based on $year(n)$ and $month(n)$, we can derive the following predicates that indicate if 2 numbers, n_1 and n_2 can represent a date, either in big endian or in little endian forms:

$$dateLE(n_1, n_2) :: month(n_1) \text{ and } year(n_2)$$
$$dateBE(n_1, n_2) :: year(n_1) \text{ and } month(n_2)$$

Finally, using these conditions, we can define the three following rules: R2 for cases where 2 numbers differ but do not correspond to a date (in which case the number on the left is assumed to be the most significant), R3 for cases where a date in little endian form is used, and R4 for cases where a date in big endian form is used.

R2 IF ($|D_{ij}| = 2$) THEN
IF (NOT (dateLE(n_{i1}, n_{i2}) AND dateLE(n_{j1}, n_{j2}))
AND NOT (dateBE(n_{i1}, n_{i2}) AND dateBE(n_{j1}, n_{j2}))) THEN
IF ($n_{i1} = n_{j1}$) THEN
IF ($n_{i2} < n_{j2}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

R3 IF ($|D_{ij}| = 2$) THEN
IF (dateLE(n_{i1}, n_{i2}) AND dateLE(n_{j1}, n_{j2}))
IF ($n_{i2} = n_{j2}$) THEN
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i2} < n_{j2}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

R4 IF ($|D_{ij}| = 2$) THEN
IF (dateBE(n_{i1}, n_{i2}) AND dateBE(n_{j1}, n_{j2}))
IF ($n_{i1} = n_{j1}$) THEN
IF ($n_{i2} < n_{j2}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

Class C

Finally, Class C corresponds to the cases where 3 numerical differences exist between the considered URIs. As for class B, it is important in that case to first detect which is the most significant of these numbers. However, we have not encountered examples other than the representations of dates using 3 numbers. Therefore, we only define the rules corresponding the dates, either in big endian or in little endian form. We define a new predicate, $day(n)$ which indicates if a number could be a day of a month:

$$day(n) :: 01 \leq n \leq 31$$

as well as the conditions to recognize dates with 3 numbers

$$dateLE(n_1, n_2, n_3) :: day(n_1) \text{ and } month(n_2) \text{ and } year(n_3)$$
$$dateBE(n_1, n_2, n_3) :: year(n_1) \text{ and } month(n_2) \text{ and } day(n_3)$$

Rules R5 and R6 corresponds to the two cases of dates with 3 numbers, in little endian and big endian forms respectively.

R5 IF ($|D_{ij}| = 3$) THEN
IF (dateLE(n_{i1}, n_{i2}, n_{i3}) AND dateLE(n_{j1}, n_{j2}, n_{j3}))
IF ($n_{i3} = n_{j3}$) THEN
IF ($n_{i2} = n_{j2}$) THEN
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$

ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i2} < n_{j2}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

R6 IF ($|D_{ij}| = 3$) THEN
IF (dateBE(n_{i1}, n_{i2}, n_{i3}) AND dateBE(n_{j1}, n_{j2}, n_{j3}))
IF ($n_{i1} = n_{j1}$) THEN
IF ($n_{i2} = n_{j2}$) THEN
IF ($n_{i3} < n_{j3}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i2} < n_{j2}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$
ELSE
IF ($n_{i1} < n_{j1}$) THEN
$[URI_i, URI_j]$
ELSE
$[URI_j, URI_i]$

4.4.3 Experiment and Evaluation

OVD is based on particular patterns which have been identified by manually analyzing Watson's repository and are not formally specified. Consequently, an empirical evaluation is crucial to verify the correctness (i.e., the precision) of OVD. In addition, such an evaluation gives us an insight on the way URIs are effectively used to encode version information, and on how we could improve OVD.

4.4.4 Experiment data

Here, we experiment with 4 sets of ontologies of increasing sizes extracted from the Watson collection (see Table 4.1). Queries 1 to 3 correspond to the OWL ontologies returned with the queries "student", "man", and "person" respectively. Query 4 corresponds to a set of nearly 7000 OWL ontologies not corresponding to any particular query.

We ran the OVD algorithm on these 4 sets of ontologies to discover versioning links between the ontologies they contain, separating the results from the different rules, in order to allow evaluating each rule separately[3]. Table 4.2 shows the number of pairs of ontologies which OVD has detected a versions of each other (see 3rd

[3]It is worth mentioning that this computation took under 5 minutes on a Mac Laptop

QueryName	QueryPhrase	Number of Ontologies
Query1	Student	90
Query2	Man	115
Query3	Person	875
Query4	*	6898

Table 4.1: Queries and corresponding ontology collections.

QueryName	Rule	Detected Pairs	Number Of Chains
Query1	R1	16	5
	R2	0	0
	R3/5	0	0
	R4	0	0
	R6	0	0
Query2	R1	16	6
	R2	0	0
	R3/5	0	0
	R4	11	5
	R6	0	0
Query3	R1	41	10
	R2	4	3
	R3/5	0	0
	R4	10	2
	R6	0	0
Query4	R1	17334	511
	R2	138119	843
	R3/5	0	0
	R4	38	7
	R6	10	4

Table 4.2: Results of running OVD on sub-sets of the Watson collection of ontologies.

column) for each set of ontologies and rules. In total, 155,589 pairs of ontology potential ontology versions have been detected. As can be seen, patterns corresponding to R1 and R2 seem to be the most applied to represent version information, with a clear different for R2 in the case of Query 4. We can also remark that R3 and R5 were never triggered in our datasets. One corresponds to dates of the form mm/yyyy and the other, to dates of the form yyyy/mm/dd.

4.4.5 Computing chains of ontology versions

While individual versioning relations are interesting to consider, many of these relations are parts of sequences of successive versions. We define and compute such *chains* of ontologies as the connected paths in the graph formed by the versioning relations detected by OVD. More formally,

Definition 1 *Given set of pairs of ontologies* $\{(O_i, O_j)\}$*, such that its cardinality is at least two, an ontology chain is defined as a sequence of ontologies,* $O_1, O_2, ..., O_{n-1}, O_n$*, such that* O_i *is the previous version of* O_{i+1}*.*

Query	Rule	Correct Chains	Incorrect Chains	Average Length	Max Length
Query 1	R1	3	2	2.6	3
	R2	0	0	0	0
	R4	0	0	0	0
	R6	0	0	0	0
Query 2	R1	4	2	2.3	3
	R2	0	0	0	0
	R4	4	1	2.6	3
	R6	0	0	0	0
Query 3	R1	6	4	1.8	3
	R2	3	0	2.3	3
	R4	0	2	2.5	3
	R6	0	0	0	0
Query 4	R1	5	5	89.05	230
	R2	9	9	71.13	154
	R4	4	3	2.2	3
	R6	2	2	2.75	3

Table 4.3: Result of the evaluation of OVD

For example, using the abbreviation A for `http://oaei.ontologymatching.org/2004/Contest/` we can compute the chain of ontology versions: [`A/testbed_v1_1.owl`, `A/testbed_v1_3.owl`, `A/testbed_v1_4.owl`]

The 4th column of Table 4.2 shows for each rule in each dataset the number of chains of ontologies that can be computed from the result of OVD.

4.4.6 Evaluation

In order to evaluate the results of OVD, we manually checked the correctness of the chains obtained for each rule in each of our datasets. Here, we assume that if one of the versioning relation in a chain is incorrect, the entire chain is incorrect. For the dataset corresponding to Query 4, we only evaluated a sample of 10 chains in the case of R1 and 18 chains in case of R2. We chose these samples randomly from different sources, trying to get examples coming from different sites in order to evaluate different conventions.The third and fourth columns of Table 4.3 show the result of this evaluation.

Using the usual measure of precision, we can evaluate the overall performance of the OVD algorithm (51.2%), as well as the individual performance of each rule. The results are presented in Table 4.4. Looking at the incorrect results, we can draw a number of conclusions concerning the way we can improve OVD. Indeed, for example, it can seen that some of the rules provide more accurate information than others. Also, it can be seen that longer chains tend to be incorrect. One of the reasons is that many incorrect results come from automatically generated ontologies URIs, which uses numbers not to represent version information, but record numbers. Using such information, we can compute different levels of confidence for the results. Finally, successive versions of ontologies tend to be similar to each other. Using a measure of similarity can help us in sorting out the correct results from the incorrect ones. Finally, some incorrect results come from RDF documents describing dated events (e.g., ESWC2006 and ESWC2007). Checking if the ontology describes dated element can also give indication that the result should be seen as incorrect.

	Total	R1	R2	R4	R6
Precision	51.2%	50%	50%	57%	50%

Table 4.4: Precision OVD's results.

4.4.7 Conclusion and Future work

In this section, general patterns which convey ontology version information directly into their URIs have been investigated, in the context of large ontology repositories such as Watson. In particular, we have identified 6 patterns which have been formalized by 6 rules. Informally, those rules describe regular sequence of characters discernible as part of the URI which hold the version information and can be used to derive versioning relations between ontologies.

Based on these rules, we described and designed OVD (Ontology Versioning Detector), which is an algorithm for detecting different versions of ontologies in large ontology repositories. It is based on two main steps: the *Selector* which compares URIs to extract sets of numerical differences between them, and the *Recognizer* which identifies the well-known pattern and try to figure out which ontology comes first. OVD has been evaluated over Watson's ontology collection, providing useful and relevant results. Indeed, the information derived from this algorithm helps us to understand how the versioning information is encoded in URIs and how ontologies evolve on the Web, ultimately supporting users in better exploiting the content of large ontology repositories. The current implementation of OVD detects different versions of the ontologies when the versioning information is expressed by single number or date-pattern. In other case OVD does not detect.

We want to extend this work considering different directions. First, new patterns not exclusively based on numbers could be detected, such as "October-2006"; new patters based on more than four numbers. Second, according to [KF01] the modification of an ontology can lead to a new version which is completely different from the original one. Although in practice, by analyzing Watson's ontology repository, we can see that it is very likely fro 2 versions of the same ontology to be similar. We can use such information to increase the precision of OVD. Finally, we can also exploit explicit version information encoded using OWL primitives and consider the overlap with information encoded in URIs.

4.5 Agreement and Disagreement

Ontologies are knowledge artifacts representing particular models of some particular domains. They are built within the communities that rely on them, meaning that they represent consensual representations inside these communities. However, when considering, like in the case of Watson, the set of ontologies distributed on the Web, many different ontologies can cover the same domain, while being built by and for different communities. Knowing which ontologies agree or disagree with others can be very useful in many scenarios.

One way to detect whether there is a disagreement between two ontologies is to rely on the presence of logical contradictions. The two ontologies can be merged, based on mappings between their entities, and the resulting model be checked for inconsistencies and incoherences. While this approach would certainly detect some forms of disagreement, it only checks whether the ontologies disagree or not. It does not provide any granular notion of disagreement and, if no contradictions are detected, it does not necessary means that the ontologies agree. Indeed, while two ontologies about two completely different, non overlapping domains would certainly not disagree, they do not agree either. More importantly, logical contradictions are not the only way for two ontologies to disagree. Indeed, there could also be conceptual mismatches, like in the case where one ontology declares that "Lion is a subclass of Species" and the other one indicates that "Lion is an instance of Species". Even at content level, logical contradictions would not detect some form of disagreements. Indeed, the two statements "Human is a subclass of Animal" and "Animal is a subclass

of Human" do not generate any incoherence. However, they disagree in the sense that, if put together, they generate results that were not expected from any of the two ontologies.

For these reasons, in [d'A09], we defined two basic measures for assessing agreement and disagreement of an ontology O with a statement $s =< subject, relation, object >$:

$$agreement(O, s) \rightarrow [0..1]$$
$$disagreement(O, s) \rightarrow [0..1]$$

We chose to use two distinct measures for agreement and disagreement so that an ontology can, at the same time and to certain extents, agree and disagree with a statement. These two measures have to be interpreted together to indicate the particular belief expressed by the ontology O regarding the statement s. For example, if $agreement(O, s) = 1$ and $disagreement(O, s) = 0$, it means that O fully agrees with s and conversely if $agreement(O, s) = 0$ and $disagreement(O, s) = 1$, it fully disagrees with s. Now, agreement and disagreement can vary between 0 and 1, meaning that O can only partially agree or disagree with s and sometimes both, when $agreement(O, s) > 0$ and $disagreement(O, s) > 0$. Finally, another case is when $agreement(O, s) = 0$ and $disagreement(O, s) = 0$. This basically means that O neither agrees nor disagrees with s, for the reason that it does not express any belief regarding the relation encoded by s.

The actual values returned for both measures, when different from 0 and 1, are not very important. They correspond to different levels of dis/agreement and only an order between pre-defined levels is needed to interpret them. The values we use and the ways to compute them are given in [d'A09].

Considering that ontologies are made of statements, extending the measures above to compute agreement and disagreement between two ontologies is relatively straightforward, using the mean of each measure for each statement of an ontology against the other ontology, in both directions and making this a normalized measure. However, while relatively simples, the two measures of agreement and disagreement between ontologies provide an interesting way to obtain an overview of a set of ontologies. Indeed, we looked at the 21 ontologies returned by Watson when querying for semantic documents containing a class with the term *SeaFood* in its ID or label, and computed the agreement and disagreement measures for all pairs of ontologies in this set. The results are shown in Figure 4.3 where ontologies are numbered according to their rank in Watson (valid on the 20/09/2009).

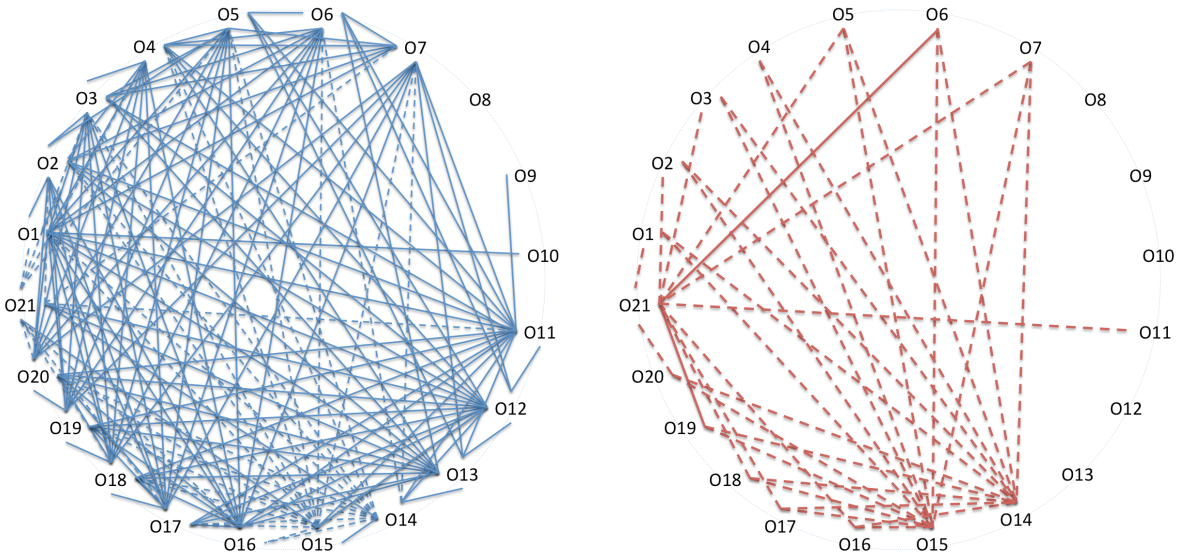

Figure 4.3: **Agreement (left) and disagreement (right) relations among the 21 test ontologies. Plain lines represent full dis/agreement (measures' values = 1). Dashed lines represent partial dis/agreement (measures' values greater than 0).**

Chapter 5

Conclusion and Future Work

Semantic Web Search Engines provide access points to the knowledge formalized on the Web. One common missing aspect in these systems regards the fact that they do not provide enough support for selecting the "right" or the "best" ontologies on the Semantic Web as shown in (Chapter 1). SWSEs have been developed without taking into account the fact that new knowledge is often produced in relation with knowledge already formalized and made available online. One of the consequences of producing knowledge in an uncontrolled way is that implicit relations between the new knowledge and the old one is not made explicit. By not making explicit these implicit relations between ontologies, SWSEs lack efficient support for applications intending to exploit multiple, interrelated ontologies on the SW.

Therefore, the main research directions of this work can be summarized as follows: 1) Studying how the ontologies can be related to each other on the SW; 2) Studying how to make explicit semantic relations between ontologies within the SWSEs.

With respect to the first direction we have been exploring research results related to the topics such as *ontology comparison* and *ontology versioning* (Chapter 2). Most of these studies focus on particular relations: *versioning relations*. The outcomes of these research regard to methods for compare two ontologies at different levels. In particular, there are approaches comparing ontologies at structural level such as **PROMPTDiff** [NM02], **OntoView** [KFKO02] and part of *SemVersion* [Vol06]. Instead, **SemVersion** [Vol06] and **OWLDiff** `http://semanticweb.org/wiki/OWLDiff` compare ontologies at a semantic level. Unfortunately, provided approaches are very limited in terms of the languages they can work with and their work in hypothesis which is the two ontologies are already related to each others. In large ontology repositories this hypothesis is not satisfied. For this reason, these mechanisms are insufficient in scenarios like the one of SWSEs. Furthermore, the previously mentioned studies do not cover most of relations we are interested in. A first theoretical result on an analysis of some relations among ontologies was done by the authors of [KA05]. They characterize a number of relations between ontologies giving abstract definitions and providing properties for all of them. We want to extend this work introducing an semantic structure DOOR to formalize concretely these and other kinds of relations. Furthermore, for all of them we are providing a concrete definition and efficient methods for detection.

Researching along the second direction, which is making explicit these implicit semantic relations between ontologies in the context of SWSEs, there are not many studies in the literature covering this topic. While designing AKTiveRank [ABS06], Alani and his colleagues have also observed that online ontologies, in particular the ones accessible through Swoogle, are often local copies of other ontologies. This raises some obvious conceptual issues when dealing with the ranking of these ontologies: the same ontology should not have been ranked at several different places in the result set. For this reason, AKTiveRank applies a very simple duplication detection procedure, syntactically comparing the ontologies. In this way, they can discover a big amount of duplications, but would miss sophisticated duplication situations, like the case of an ontology formalized in different ontology languages (e.g.,iswc.owl and iswc.daml). Although the main research results discussed so far are very important as starting point for this work, there are different aspects and different relations that still need to be addressed in context of SWSEs, where there are a huge number of ontologies

to deal with.

In order to achieve our goal, we propose KANNEL an ontology based framework to detect and manage relations between ontologies in large ontology repositories. It is based on the DOOR ontology which has the aims to provide the missing interpretation of the existing semantic relations among the ontologies. The advantages of using KANNEL are that:

- It is very flexible. At any time we need to deal with a new relation, all that we need is to add its formalization to the DOOR semantic structure and design a method to detect it;
- It is possible to apply it to any ontology repository, of any current SWSE.
- Ongoing maturation of Semantic Web Technologies makes easier the implementation of the mentioned approach, because the needed tools are now available and reusable, and evaluation is made easier in concrete scenarios such as Watson.

KANNEL will also provide particular mechanisms to navigate through the ontologies in a structured way, that is following the satisfied relations among them. Furthermore, by this framework we are able to acquire an overall understanding of the growth of the SW in terms of its content, structure and evolution, which is important for facilitating the development of the SW in a controlled and informed way.

As regards future work, in the short term, we need to carry on with the development of detection mechanisms for ontology relations, as well as with the continuous development of the DOOR ontology. Until now, our focus has been on the off-line part of the KANNEL framework, populating the DOOR ontology with discovered relations. The next step concerns the development of a reasoner-based API for exploring and querying the discovered relations on the basis of both ontology and rule-based reasoning. This API will be used as the basis to integrate the features of DOOR into Watson to provide relationship information together with the search results. Thanks to this integration, we will be able to evaluate the impact of such features in helping users in selecting ontologies.

Bibliography

[ABS06] H. Alani, C. Brewster, and N. Shadbolt. Ranking ontologies with `AKTiveRank`. *5th Int. Semantic Web Conf. Athens, Georgia, USA*, 2006.

[AdM09] C. Allocca, M. d'Aquin, and E. Motta. Door: Towards a formalization of ontology relations. *Proc. of the Inter. Conf. on Knowledge Engineering and Ontology Development (KEOD)*, 2009.

[All09] Carlo Allocca. Making explicit semantic relations between ontologies in large ontology repositories. *PhD Symposium, Poster Session, ESWC*, 2009.

[Art] Alexander S. Kleshchev Irene L. Artemjeva. Unenriched logical relationship systems.

[BQL07] David Bell, Guilin Qi, and Weiru Liu. Approaches to inconsistency handling in description-logic based ontologies. *Proc of the SWWS Conference.*, 4825/2008, 2007.

[d'A09] M. d'Aquin. Formally measuring agreement and disagreement in ontologies. *5th K-CAP*, 2009.

[DE08] J. David and J. Euzenat. Comparison between ontology distances (preliminary results). *7th Int. Semantic Web Conference, ISWC*, 2008.

[dMea08] M. d'Aquin, E. Motta, and M. Sabou et al. Towards a new generation of semantic web applications. *IEEE Intell. Sys.*, 23(3), 2008.

[dSD+07] M. d'Aquin, M. Sabou, M. Dzbor, C. Baldassarre, L. Gridinoc, S. Angeletou, and E. Mottta. Watson: A gateway for the semantic web. *Poster Session at 4th ESWC*, 2007.

[GGMO01] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Understanding top-level ontological distinctions. 2001.

[GHKS07] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the right amount: Extracting modules from ontologies. In *Proceedings of WWW*, pages 717–726, Banff, Canada, 2007. ACM.

[GLW06] S. Ghilardi, C. Lutz, and F. Wolter. Did I damage my ontology? a case for conservative extensions in description logics. In *Proc. of the 10th Inter. Conf. on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 187–197. AAAI Press, 2006.

[GPS99] A. Gangemi, D. M. Pisanelli, and G. Steve. An overview of the onions project: Applying ontologies to the integration of medical terminologies. *Technical report. ITBM-CNR, V. Marx 15, 00137, Roma, Italy*, 1999.

[HA06] N. Shadbolt H. Alani, C. Brewster. Ranking ontologies with aktiverank. *Proc of the 4th Int Sem Web Conf (ISWC2006), 2006, LNCS, Springer*, pages 1–15., 2006.

[Hef01] J. Heflin. Towards the semantic web: Knowledge representation in a dynamic, distributed environment. *Ph.D. Thesis, University of Maryland, 2001*, 2001.

[HP04] J. Heflin and Z. Pan. A model theoretic semantics for ontology versioning. *3th Intern Sem Web Conf, Hiroshima, Japan, LNCS 3298 Springer*, pages 62–76., 2004.

[HPea05] J. Hartmann, R. Palma, and et al. Ontology metadata vocabulary and applications. pages 906–915, OCT 2005.

[HS05] Z. Huang and H. Stuckenschmidt. Reasoning with multi-version ontologies: a temporal logic approach. *Proc of the Fourth Intern Sem Web Conf (ISWC2005),3729, LNCS, Springer*, pages 62–76., 2005.

[KA05] A. Kleshchev and I. Artemjeva. An analysis of some relations among domain ontologies. *Int. Journal on Inf. Theories and Appl*, 12:85–93, 2005.

[KCDF08] B. Konev, C.Lutz, D.Walther, and F.Wolter. Cex and mex: Logical diff and logic-based module extraction in a fragment of owl. *Liverpool University, UK and TU Dresden, Germany*, 2008.

[KF01] M. Klein and D. Fensel. Ontology versioning on the semantic web. *Proc. of the Inter. Semantic Web Working Symposium (SWWS)*, pages 75–91, 2001.

[KFKO02] M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. Ontology versioning and change detection on the web. *13th Intern Conf on Know. Engineering and Know. Management (EKAW02)*, pages 197–212, 2002.

[MS02] A. Maedche and S. Staab. Comparing ontologies-similarity measures and a comparison study. *Proc. of EKAW-2002*, 2002.

[MSM06] Vanessa Lopez Marta Sabou and Enrico Motta. Ontology selection for the real semantic web: How to cover the queenÕs birthday dinner? *Proc. of EKAW-2006*, 2006.

[NM02] N. F. Noy and M. A. Musen. Promptdiff: A fixed-point algorithm for comparing ontology versions. *18th National Conf. on Artificial Intelligence (AAAI)*, 2002.

[PSHH04] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. Owl web ontology language semantics and abstract syntax. *W3C Recommendation*, 2004.

[QH07] Guilin Qi and Anthony Hunter. Measuring incoherence in description logic-based ontologies. *Proc of the Int. Sem. Web Conference.*, 4825/2008:381–394, 2007.

[SPd08] E. Motta S. Peroni and M. d'Aquin. Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5367/2008:242–256, 2008.

[Vol06] M. Volkel. *D2.3.3.v2 SemVersion Versioning RDF and Ontologies. EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB.* 2006.

[Wel05] Katrin Weller. An analysis of some relations among domain ontologies. *International Journal: Informations Theories and Applications. Vol 12*, 2005.

[Wel08] Katrin Weller. Koso. a reference-ontology for reuse of existing knowledge organization systems. *Proceedings of the 1st Workshop on Knowledge Reuse and Reengineering over the Semantic Web (KRRSW 08), ESWC 2008*, pages 31–40, 2008.