國立臺灣大學管理學院資訊管理學研究所

碩士論文

Department of Information Management

College of Management

National Taiwan University

Master Thesis

關聯式資料庫至知識本體之對映:
一套結合豐富語意與對映一致性的方法
Mapping Relational Databases to Ontologies: An Approach Combining Semantic Enrichment and Mapping Consistency

劉俊宏

Jun-Hong Liu

指導教授：蔡益坤 博士

Advisor: Yih-Kuen Tsay, Ph.D.

中華民國 97 年 7 月

July, 2008

致 謝

時間過得很快，隨著論文的結束，轉眼間兩年的研究生生涯就即將劃下句點，驚呼時光居然是一眨眼的就過去了，雖有不捨，但心中更多的是滿滿的感謝與喜悅，感謝這兩年來許多人的幫助與鼓勵，以及對論文可順利付梓的無盡感動。回想起這兩年，對於此論文的相關領城知識，從懵懂無知到略有所懂，最後可以將論文完成，這之中，過程雖然艱辛但卻獲益良多。

本論文能夠順利完成，要感謝的人很多。首先最要感謝的就是我的指導老師-蔡益坤老師，兩年多來不辭辛勞地費心指導，使我學會從事研究的正確態度與方法；以及對我論文耐心地幫助、指正，口試前不厭其煩地對報告再三叮嚀，使我在口試過程中得以劃下完美的句點，實有無盡的感謝。也要感謝口試委員老師-陳建錦老師、李瑞庭老師，在口試期間不吝指導與斧正，給予許多寶貴的意見及指教，使本論文能更臻完整而嚴謹。

另外，也相當感謝實驗室一起打拼兩年的同班同學們，德威、志華、文振、啟建，一起熬夜趕作業、念 paper、寫程式、互相打氣，給予我論文許多協助與意見，陪我走過這段非常忙碌卻也相當充實的研究生活，你們真的是最好的夥伴；也感謝實驗室的學弟妹，依珊、正一、晉碩、敔傑，實驗室因為有你們而增添了不少歡樂，你們也是功不可沒的一群;也要感謝實驗室得高望重的博班學長，郁芳、明憲，解決我許多課業上的問題，有你們在真是家有一老如有一寶。

最後，要感謝的就是默默在背後支持我的家人及女友；尤其是最親愛的爸爸、媽媽，在我求學生涯中給我無盡的支持，讓我可以心無旁騖地在學業上衝刺，以及爺爺、奶奶、姐姐們也時常透過電話關心我的生活，給予我最溫暖的依靠。最後就是感謝女友思佳陪我一起度過論文生產的過程，在我挫折時給我最大的支持與鼓勵，是我精神上重要的依靠。

僅以此論文獻給我的親人與所有關心我的人，謝謝你們！

劉俊宏　謹致
于台大資訊管理研究所
民國九十七年七月

論 文 摘 要

作者：劉俊宏　　　　　　　　　　　　　　　　　　　　　　　　九十七年七月

指導教授：蔡益坤 博士

關聯式資料庫至知識本體之對映:
一套結合豐富語意與對映一致性的方法

語意網的核心挑戰之一就是將現有的大量資訊轉變成由知識本體語言所定義的知識本體，這些現有的大量資訊主要是由網路上的資訊內容所構成，因此，為了實現語意網的目的，將這些網路上的資訊內容對映至知識本體是有必要的。既然網路上擁有超過數億的網頁，而這些網頁上的資訊內容大多數是被儲存在關聯式資料庫因而很難被搜尋引擎所找到(俗稱“深網”)。因此，為了讓語意網可以取用這些資訊內容，一個有效的方法就是將深網底下的關聯式資料庫對映至某特定領域中一個已有的知識本體。

在這篇論文中，我們設計出一套半自動化方法直接將一個關聯式資料庫對映到某特定領域中一個已有的知識本體，這個方法採取在資料採礦中的群集分析概念去替每個表格找出它們的匹對類別群組，其中在類別群組中的每個類別都會滿足對映一至性亦即對映的結果不應該違背在關聯式資料庫所表達的事實。我們將此方法分成兩階段。第一階段主要是利用語意上的資訊將關聯式資料庫中的外鍵對映到知識本體中的物件屬性以獲得一些表格的匹對類別群組特徵，有別於其它方法，我們在對映時,不只考量了表格之間的一般化/特殊化關係也考量了隱含表達在關聯式資料庫中的反關係以建構出一個隱含匹對等級。第二階段使用外鍵的對映結果作為特徵線索以找出一些表格的匹對類別群組，然後再利用這些表格的匹對類別群組去找到其它表格的匹對類別群組，最後將每個表格對映至本身的匹對類別群組。一個雛形化系統顯示出利用此方法執行一些現實世界中的樣本對映有良好的績效。

關鍵字：語意網，知識本體語言，深網，對映，關聯式資料庫，群集分析，對映一致性

THESIS ABSTRACT
GRADUATE INSTITUTE OF INFORMATION MANAGEMENT
NATIONAL TAIWAN UNIVERSITY

Student: Liu, Jun-Hong
Advisor: Tsay, Yih-Kuen

Month/Year: July, 2008

Mapping Relational Databases to Ontologies: An Approach Combining Semantic Enrichment and Mapping Consistency

One of the core challenges of the Semantic Web is to transform mass existing information to OWL ontologies. This mass existing information is mainly composed of the contents on the Web. Therefore, to realize the Semantic Web, it will be necessary to map the Web contents to OWL ontologies. Manually mapping the Web contents to OWL ontologies is impractical since the Web has well over billions of Web pages and most of the contents of Web pages is stored in the relational databases and hence hard to be found by search engines (so-called the "*deep Web*"). Hence, to make these contents available for the Semantic Web, an effective way is to map the relational databases underlying the deep Web to domain-related OWL ontologies.

In this thesis, we propose a semi-automatic approach for directly mapping relational databases to OWL ontologies. This approach takes the concept of cluster analysis in data mining to find the matching classes group (MCG) for every table, where every found class of MCG will satisfy the mapping consistency stating that the mapping results should not violate the fact expressed in the relational database. We divide our approach into two phases. The first phase primarily uses the semantic information to map the foreign keys of a relational database to object properties of OWL ontologies to get the features of MCGs for tables. Different from other approaches, we not only take into consideration the Generalization/Specialization relationship between tables but also take advantage of the inverse relationship between tables which are implicitly expressed in a relational database to construct an implicit matching level between foreigns keys and object properties. The second phase uses the mapping results of the foreign keys as clues of features to find MCGs for some tables and then takes the MCGs of these tables to find MCGs for other tables. Finally, every table is mapped to its MCG. A prototype system demonstrates that our approach performs well on several domain samples from the real world.

Keywords: Semantic Web, OWL, Deep Web, Mapping, Relational Databases, Cluster Analysis, Mapping Consistency

Contents

1 Introduction **1**
- 1.1 Background 1
- 1.2 Motivation and Objectives 2
- 1.3 Thesis Outline 3

2 Related Work **5**
- 2.1 Semantic Web 5
  - 2.1.1 Ontology 6
- 2.2 Deep Annotation 7
- 2.3 Mapping between Relational Databases and OWL Ontologies 9
  - 2.3.1 Extracting OWL ontologies from Relational Databases 9
  - 2.3.2 Mapping Relational Databases to Existing OWL Ontologies . . . 11
- 2.4 Matching Techniques 13
  - 2.4.1 String-Based Matching 13
  - 2.4.2 WordNet 14
  - 2.4.3 Graph Matching 15
  - 2.4.4 Matching Systems 16

3 Preliminaries **20**
- 3.1 Relational Database 20
  - 3.1.1 Relational Data Model 21
  - 3.1.2 Relational Database Normalization 23
  - 3.1.3 Entity-Relationship model 24
- 3.2 OWL: Web Ontology Language 24
  - 3.2.1 OWL Lite 25
  - 3.2.2 OWL DL and OWL Full 28
  - 3.2.3 The Mapping Between OWL DL And Description Logic 29
- 3.3 Differences Between Database Schema and Ontology 30

4 Mapping Approach **32**
- 4.1 Overview of the Approach 33
- 4.2 Preliminary Definitions 35
- 4.3 Classifying Tables and Columns 36
  - 4.3.1 Classifying Tables 37
  - 4.3.2 Classifying Foreign Keys 39
- 4.4 Matchers in the Approach 40

4.4.1 Matchers 40
4.4.2 Source of Computing the Similarity 43
4.5 Mapping Foreign Keys to Object Properties 44
4.5.1 Mapping Base FKs 44
4.5.2 Mapping Part-Of FKs 46
4.5.3 Mapping FKs of Relationship Tables 47
4.6 Mapping Tables 48
4.6.1 MCGs of Associated Tables 48
4.6.2 MCGs of G/S Tables and Other Tables 50
4.6.3 Mapping Tables and Non-Foreign Keys 52
4.6.4 Mapping Columns 53

5 Prototype System: Annotator **57**
5.1 Annotator 57
5.1.1 Matcher 58
5.1.2 Relational Database 59
5.1.3 OWL Ontology 59

6 Conclusion **64**
6.1 Contributions 64
6.2 Future Work 66

List of Figures

2.1 Semantic Web Layers 6
2.2 An Architecture for Deep Annotation 8
2.3 The GLUE Architecture 18
2.4 Match processing in COMA 19

4.1 System Architecture 34

5.1 System Architecture of The Traveller 61
5.2 Annotator 62
5.3 Example: Relational Database Schema 63

List of Tables

3.1 OWL Constructors and Corresponding Description Logic Syntax 30

4.1 Classification of Tables and Columns 35
4.2 Groups of SQL and XML Schema Datatypes 55

Chapter 1

Introduction

1.1 Background

The information on the World Wide Web primarily consists of Web pages written in HTML (Hyper Text Markup Language), a *markup* language that displays the information on the Web. However, most of the information written in this format is only for human consumption. As a consequence, computers or software agents can not understand and process this kind of information effectively.

In order to make the computers exchange and process this kind of information effectively, one way is to provide the information in such a way that computers can understand it. For this purpose, Tim-Berners Lee presented the Semantic Web [7] , "*The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.*"

The dream of the Semantic Web can be realized through using a well-defined structure data such as *ontology*. An ontology is a formal, explicit specification of a shared conceptualization [17]. Recently the World Wide Web Consortium (W3C) has developed the *Web Ontology Language* (OWL) [29] providing a language defining the structured, Web-based ontologies which allow a richer integration and interoperability of data among communities and domains. With the OWL standard, searching, sharing and reusing information between Webs will be easier and more reliable. Nevertheless, currently the mass existing information is mainly composed of the relational databases and the resources on the Web and this kind of information has the heterogeneity with OWL ontologies. Besides, most of the resource on the Web are stored in the relational databases (so-called the "*deep*

Web") , and they are hard to be found by search engines. For this reason, to achieve interoperability between the existing information and OWL ontologies, an effective way is to create mapping between relational databases and OWL ontologies.

1.2 Motivation and Objectives

As mentioned above, most of the Web contents are stored in relational databases, and they usually represented by *dynamic pages*. The Web pages are primarily composed of static and dynamic pages. Static pages are based on HTML formats and not constantly updating. In contrast to dynamic pages, the contents of the static pages are not stored in databases but simply placed on a server.

Dynamic pages has rapidly increased in recent years with the development of the Web technologys such as the Web 2.0. Most of the data on dynamic pages is created as the result of a database retrieval. These data represent the invisible Web, ignored by search engines. For instance, on the Web "Amazon.com", all of the information such as books and cost are stored in databases, these information can be seen only by users who visit the Web but can not be fully found by search engines like Goggle and Yahoo.

In order to make the contents in databases available for the Semantic Web, Handschuh et al. proposed a framework "deep annotation" [18] , which is an annotation process that creates metadata from existing information to derive mappings between information structures such as ontologies and databases. This process has three main parties and each needs human participation 2.2. For instance, the *annotator* in this framework maps the Web to his ontologies by using simple heuristics and translation rules to examine the HTML or the underlying database.

However, the number of the Web pages has dramatically increased, manual annotation of HTMLs or databases is not practical. In order to make the best of these underlying databases and to realize the requirement for the Semantic Web, mapping between databases and OWL ontologies is necessary. The mapping process is to find the semantic correspondences between entities or elements of different information structures. Since the emergence of the Semantic Web and OWL, many studies have been concerning about mapping between OWL ontologies. Nevertheless, on the other hand, literatures in mapping between databases and OWL ontologies are rare.

Mapping between databases and OWL ontologies can be divided into two parts. The one is to extract OWL ontologies from existing databases and the other is directly to map databases to existing OWL ontologies. We believe that using the former one, in the long run the generated OWL ontologies from databases will be mapped to the existing domain-related OWL ontologies in order to reach the interoperateliablity between them, and this will lose more semantic information comparing with the original database. Besides, considering the Web contents are primarily stored in the relational database, and hence the object in this thesis is to propose a approach that directly maps a relational database to domain-related OWL ontologies.

1.3 Thesis Outline

Now, we briefly describe the remaining chapters of this thesis as follows:

In Chapter 2, first, we introduce the Semantic Web, especially for the *ontology*. Second, due to the problem of the deep Web the *deep annotation framework* is described. The goal of this framework is to propose a solution to map or migrate the databases behind the deep Web to the ontologies. Third, we introduce two main approaches of *Mapping between Relational Databases and OWL ontologies* including *Extracting OWL ontologies from Relational Databases* and *Mapping Relational Databases to Existing OWL Ontologies*. The latter case is our objective in this thesis. Finally, we describe some matching techniques that assist greatly the mapping between databases and ontologies.

In Chapter 3, since our object is to map a relational database to existing OWL ontologies we first examine the basic concept of Web Ontology Language (OWL) and relational database. Then the difference between databases and ontologies are described. Finally since OWL is based on the Description Logics (DLs) we also illustrate the mapping between OWL DL and DLs.

In Chapter 4, we introduce our approach to directly mapping a relational database to OWL ontologies. We first describe our terms and functions of the relational database and OWL ontologies used in our approach. Then the detailing of approach is discussed.

In Chapters 5 and 6, a mapping system "Annotator" implementing the approach proposed in this thesis, our contributions and future works are introduced.

Chapter 2

Related Work

In this chapter, we will introduce some related studies in mapping between a relational database and OWL ontologies. Our object is to map a relational database to domain-related OWL ontologies to realize the requirement for the Semantic Web. Thus, we begin to discuss the Semantic Web and his enabling technology *ontology*. Then the researches of the deep annotation and the database-to-ontology mapping are described. Finally, we introduce some matching techniques that do a lot of assistance in these fields.

2.1 Semantic Web

Since the development of the Web technologys, Web pages has increased rapidly. According to Google, currently the number of Web pages has well over 15.5 billion [32] . These Web pages contained all kinds of information, however, the vast majority of these information is only in a human understandable format such as HTML. More precisely format is eXtensible Markup Language(XML). Although XML provides a set of self-defined metadata tags to describe the semantic of Web data, it does not define the meaning of the tags. As a consequence, Web content can be accessed only in the syntactic level and software agents or machines can not efficiently understand and process this kind of data.

For this purpose, in 2001 [7] , Tim-Berners Lee proposed the vision of the Semantic Web as follows: *"The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation."* To realize the requirement for the Semantic Web, World Wide Web Consortium(W3C) and other organizes have been effected at specifying and developing standard language. The Semantic Web will built on the standard layers as

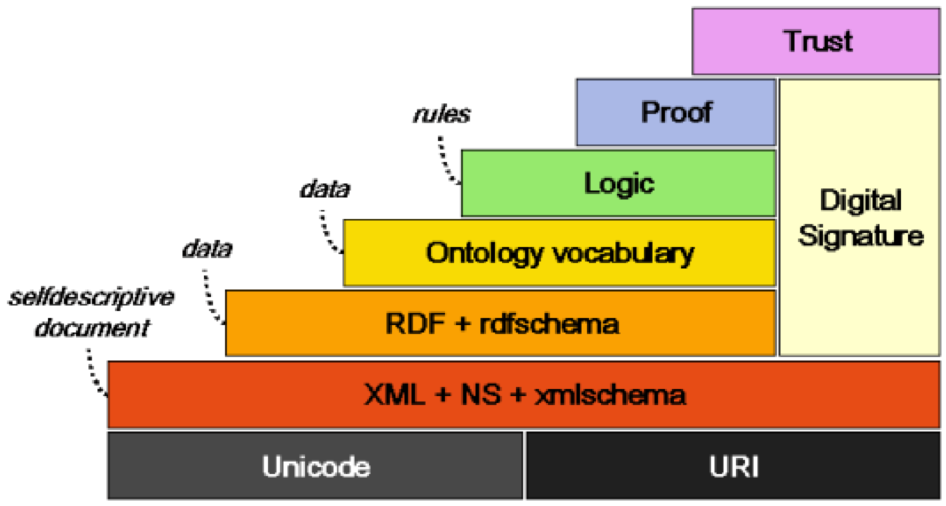


Figure 2.1: Semantic Web Layers

shown in Figure2.1 [22]. In next section, we will present ontology which is the core component in these layers.

2.1.1 Ontology

The term “Ontology” came from philosophy and there are plenty of definitions, the most popular is “An ontology is a formal, explicit specification of a shared conceptualization” [17] . Ontology is a key enabling technology for the Semantic Web. Unlike the XML, the level ontologies provides not only the syntactic but the semantic. By ontologies, we can build a controlled vocabulary of concepts, each with explicitly defined and machine-understandable semantic. Thereby, people and machines can communicate precisely. For this reason, during the last decade interest to ontologies has increased. The area of applicability for ontologies is wide: information retrieval and extraction, information systems design and enterprise integration, natural language processing, database design, conceptual modeling. In the Semantic Web area, a formal knowledge representation model is also called an ontology.

However, constructing ontologies for domain knowledge is still tedious, time-consuming and error-prone. Although some tools like Protege [38] can manually edit ontologies, fully automatic ontology building remains in the distant future. In order to make the ontology-construction efficient, Maedche and Staab presented an ontology-learning framework [27] that encompasses ontology import, extraction, pruning, refinement, and evaluation. The overview of framework is as follows :

- Merging existing structures or defining mapping rules between these structures allows *importing* and *reusing* existing ontologies.
- Ontology *extraction* models major parts of the target ontology, with learning support fed from Web documents.
- The target ontology's rough outline, which results from import, reuse, and extraction, is *pruned* to better fit the ontology to its primary purpose.
- Ontology *refinement* profits from the pruned ontology but completes the ontology at a fine granularity (in contrast to extraction).
- The target application serves as a measure for validating the resulting ontology.

Furthermore, in order to have the consistent format to model ontology, W3C had defined the standard for Web ontology that is OWL(Web Ontology Language) [29]. OWL is a language for ontology serves as defining and conceptualizing the Web content. We can conceptualize or model a domain knowledge through OWL features such as *class* and *property.* Currently, OWL is built on top of RDF(Resource Description Framework) and divided into three sub-languages depend on the expressiveness, namely OWL Lite, OWL DL and OWL Full. In Chapter 3, we have the details for OWL.

2.2 Deep Annotation

As mentioned above, Web pages is well over 15.5 billion [32] . Annotating these Web pages with OWL is a key step to reach the goal of the Semantic Web. However, there is still lack of the existing automatic tools to annotate them especially for the dynamic Web pages. Dynamic Web page is also called a *deep Web* which most of its content comes from the underlying databases. In contrast, the static Web page is called *surface Web.* A July 2000 white paper [10] estimated 43,000-96,000 deep Web sites and an informal estimate of 7,500 terabytes of data- 500 times larger than the surface Web.

However, the content of deep Web is hardly reached by search engines. For this purpose, emerging the *deep annotation* [18] . *Deep annotation* is a vision of framework

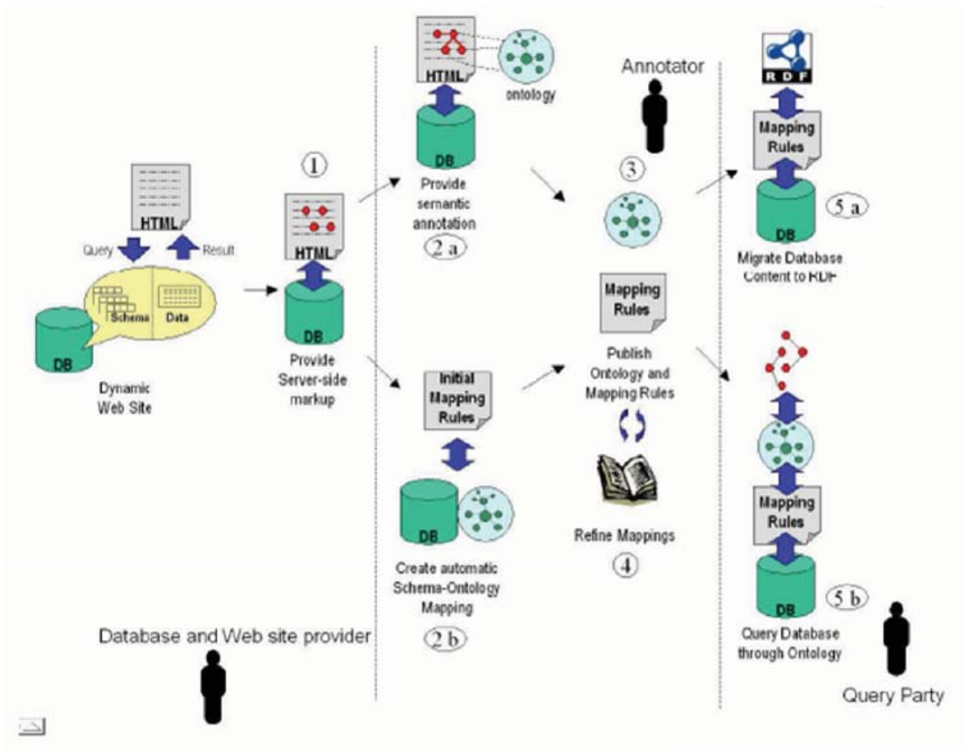

HTML
Query
Result
Schema
Data
DB
Dynamic Web Site
①
HTML
DB
Provide Server-side markup
HTML
ontology
DB
Provide semantic annotation
2 a
Initial Mapping Rules
DB
Create automatic Schema-Ontology Mapping
2 b
Annotator
3
Mapping Rules
Publish Ontology and Mapping Rules
Refine Mappings
4
RDF
Mapping Rules
DB
5 a
Migrate Database Content to RDF
Mapping Rules
DB
5 b
Query Database through Ontology
Query Party
Database and Web site provider

Figure 2.2: An Architecture for Deep Annotation

to provide semantic annotations for the underlying databases of the deep Web. This framework is composed of three parties: *Database and Web Site Provider*, *Annotator*, and *Query Party*- see Figure2.2 [18]. It assumes that many web sites will in fact participate in the Semantic Web and will support the sharing of information. Hence, Web site providers will give their users information proper, information structure and information context.

The details of the architecture are as follows:

- Database and Web site providers markup the dynamic Web pages according to the information structures of the database.
- The annotator produces client-side annotations conforming to the client ontology either via the marked HTML documents or database schemas.
- The annotator publishes the client ontology and the mapping rules derived from annotations.
- The annotator assesses and refines the mapping using certain guidelines.

- Query party mappings into his own information structures such as ontologies and/or to migrate the data into his own repository.

2.3 Mapping between Relational Databases and OWL Ontologies

Due to the requirement for the Semantic Web and the problem of the deep Web, many researches has focused on the mapping between a relational database and OWL ontologies [2] [24] [37] [1] [19]. In this field, mapping is the process to find the semantic correspondences between entities or elements of the relational database and OWL ontologies. Currently there are some solutions and tools to deal with this problem. They can be classified into two parts: methods for creating ontologies from existing relational database and methods for mapping a relational database to already existing OWL ontologies. If the former case that generated ontologies finally will be mapped to domain-related ontologies , we think it will lose more semantic information comparing to the original database. For this reason, our work will focus on the latter case. In next two sections, we will discuss these two kinds of methods.

2.3.1 Extracting OWL ontologies from Relational Databases

Extracting OWL ontologies from relational databases is a *Reverse engineering* which is the process of analyzing an existing system; identifying system components, abstractions, and interrelationships; and creating representations of them. There are plenty of work on the reverse engineering of relational databases. However, most of them focus on extracting E-R(entity-relationship) and object models from them, the semantics obtained by these methods cannot fully meet the requirement of constructing ontologies. Until recent years, there exist a few approaches that consider ontologies as the target for reverse engineering. In this section, we will introduce some studies in this field.

Astrova [2] presented a method to extract ontologies from a relational database. This method is composed of two processes: By analyzing information such as key, data and attribute correlations to extract a conceptual schema, which expresses semantics about the relational database, and transforming this schema into a semantically equivalent ontology. Finally migrating data from a database to ontologies. Transforming the relational schema into a semantically equivalent ontology proceeds as follows:

1. Classification of relations: Depend on the features, relations can be classified into three categories:
    - Base relations: If a relation is independent of any other relation in a relational database schema, it is a base relation.
    - Dependent relations: If a primary key of a relation depends on another relations' primary key, it is a dependent relation.
    - Composite relations: A composite relation is a relation that is neither base nor dependent and its primary keys are composed of other relations' primary keys.
2. Mapping: Key, data and attribute correlations can be described given two relations. Each kind of correlation has four types: *equality*, *inclusion*, *overlap* and *key disjointedness*. With the combination of these types of correlation and the development of the mapping constraint, how to extract a ontology from a relational database schema can be determined.

Since the relational database schema often has little explicit semantics [33] , through analyzing tuples in the relational database, additional "hidden" semantics (e.g. inheritance) can be discovered. However, it is very time consuming with regard to the number of tuples of the relational database.

Astrova also presented a method [3] that constructed an ontology based on analyzing the HTML-forms to extract a form model schema, transforming the form model schema into ontology and creating ontological instances from data contained in the pages. The drawback of this approach is that it does not offer any way to the identification of inheritance relationship which is a significant aspect in the ontology construction. In order to overcome this drawback, Benslimane proposed an approach [9] to acquire ontologies from data-intensive webs. The main idea of this approach is the fact that users often query database through HTML forms and the query results often return as HTML tables. Thus, the data in the HTML forms are often structural data. By analyzing a HTML-form, important information can be obtained. Besides, this method use an enriched relational schema instead of simply use the relational schemas that is constructed from database. The processes of this method are as follows:

1. Extract forms schema by analyzing HTML pages. It uses several identification rules and translation rules to identify the form unit and generate the XML-schema.
2. Restructure and enrich the relational schema through semantics of the forms schema. In this step, the result of the relational schema is mostly like the structure of the underlying database. But it has additional inclusion dependencies and constraints.
3. Construct OWL ontology from the enriched relational schema using a set of transformation rules. These rules construct classes, properties, and Inheritance from the semantic similarities between the relational schema and ontology(OWL).

Man Li [24] extracted ontology in a relational database using E-R Model. This approach defined twelve rules for extracting ontology from the relational database schema and used these rules to create ontology.

Trinh [37] proposed a tool named *RDB2ONT* that creates ontology in a relational database. This tool is a method using an ontology to describe relational database and converting the information in a relational database into this ontology.

2.3.2 Mapping Relational Databases to Existing OWL Ontologies

In this section we will discuss some approaches that directly mapping relational databases to OWL ontologies. These approaches are assumed that domain-related ontologies and legacy relational databases already exist. Because there is a different domain level or size between databases and ontologies, and the modeling criteria used for designing databases is also different from those used for designing ontology models. Thus, compare with the approaches that extract ontologies from databases, mapping approaches here are rare and more complex.

Borgida et al. [1] proposed a method that assists users in specifying and inferring mapping formulas between relational databases and OWL ontologies. Based on the simple correspondences between relational database schemas and OWL ontologies, complex formulas expressing the semantic mapping can be found. However, this method has a disadvantage that the database must be based on ER design principles and cannot extract classes that can be separated in the fields within a table. Besides, since only scheme is taken account of without consideration on instance, it is hard to match precisely.

Hu and Qu presented an approach [19] that uses virtual documents based on TF/IDF model to discover simple mappings between the relational database schema and OWL ontologies. Besides, this approach also finds the subsumption relationships, called contextual mappings, which can be directly translated to conditional mappings or view-based mappings [11] . The overview of this approach is as follows:

1. Classifying entity types: This is a preprocessing process. It classifies entities into the relational schema and the ontology into four different groups to limit the searching space of candidate mappings. Besides, it coordinates different characteristics between the relational schema and the ontology.

2. Discovering simple mappings: This step firstly constructs virtual documents for the entities in the relational schema and the ontology to capture their implicit semantic information. Then, it discovers simple mappings between entities by calculating the confidence measures between virtual documents via the TF/IDF model

3. Validating mapping consistency: This phase uses mappings between relations and classes to validate the consistency of mappings between attributes and properties. It considers the compatibility between data types of attributes and properties as well. In addition, some inference rules are also integrated in this process.

4. Constructing contextual mappings: This phase operates on mappings between relations and classes found in the previous phases, and supplies them with sample instances. It constructs a set of contextual mappings, which indicate the conditions how they could be transformed to view-based mappings with selection conditions.

Furthermore, there exist some studies that deal with the other problems from other aspects. For instance, Dou et al. [15] describe a general framework for integrating databases with ontologies via a first-order ontology language Web-PDDL. Barrasa et al. [5] design a language R2O to express complex mappings between relational database schemas and ontologies.

2.4 Matching Techniques

After introducing the mapping between relational databases and OWL ontologies, we now discuss some matching techniques that do a lot of help in this field. Matching is the process that takes two data structures such as schemas or ontologies as input and calculates similarity relationship between their entities or elements as output. It plays a important role in many application domains, such as semantic web, schema/ontology integration, data warehouses, e-commerce, query mediation, information retrieval, etc. Many matching approaches have been proposed so far, and in this section we will introduce some of them include string-based matching, WordNet-based matching, graph-based matching and some matching systems.

2.4.1 String-Based Matching

String-based matching is a element level matching that takes the information such as local descriptions of two elements as input and return their similarity measure as output. This technique is widely used in various schema matching systems [26] [13] . In this section, some popular string-based matchers are discussed.

- Edit-distance: It determine the distance between two strings based on the minimal number of edit operations (insert, delete, replace) needed to transform one string into the other. There are various edit-distance matchers, for instance, a famous matcher called *Levenstein distance* [23] is the minimum number of insertions, deletions, and substitutions of characters required to transform one string into the other.

- Jaro measure: The Jaro measure has been defined for matching proper names that may contain similar spelling mistakes [20] . It is not based on an edit distance model, but on the number and proximity of the common characters between two strings. The definition of Jaro is as below: Let $s[i] \in com(s,t)$ if and only if $\exists j \in [i - (min(|s|,|t|)/2, i + (min(|s|,|t|)/2]$ and $transp(s,t)$ are the elements of $com(s,t)$ which occur in a different order in s and t.

$$Jaro_{\sigma(s,t)} = \frac{1}{3} \times \frac{|com(s,t)|}{|s|} \times \frac{|com(t,s)|}{|t|} \times \frac{|com(s,t)|\,|transp(s,t)|}{|com(s,t)|}$$

- N-gram similarity: The N-gram similarity is also often used in comparing strings. It computes the number of common n-grams, i.e., sequences of n characters, between them. For instance, three-grams for the string article are: art, rti, tic, icl, cle. Its definition is: Let ngram(s, n) be the set of substrings of s of length. The n-gram similarity is a similarity σ such that:

$$n - gram_{\sigma(s,t)} = \frac{|ngram(s,n) \bigcap ngram(t,n)|}{min(|s|\,,|t|) - n - 1}$$

- Token-based distances: This technique come from information retrieval and consider a string as a (multi)set of words (also called bag of words). A very common measure is TFIDF, many systems use measures based on it. In this model, each string is represented as a vector containing weights for each term(word) of a global corpus. The similarity of two strings is then determined with the cosine measure.

2.4.2 WordNet

String-based matchers can only reach the syntactic matching. In order to get to the semantic matching, we need some other tools such as WordNet. WordNet [31] is a large lexical database of English, words are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Each synset has a gloss that defines the concept that it represents. Synsets are connected each other through explicit semantic relations such as hypernymy, hyponymy for nouns and hypernymy and troponymy for verbs. These relations constitute kind-of and part-of hierarchies. For example, the word *author* has a *Synonym writer*, and they constitute a *sysset* which has a *gross* "writes (books or stories or articles or the like) professionally (for pay)" to describe it. The word *communicator* is the *hypernymy* of this *synset*, hence, *author*, *writer* and *communication* constitute a hierarchy relation. Thereby, a matcher based on WordNet can be designed by the following rules: Given two words s and t,

- $t \sqsubseteq s$, if t is a hyponym or meronym of s

- $t \sqsupseteq s$, if t is a hypernym or holonym of s
- $t \equiv s$, if they are connected by synonymy relation or they belong to one synset.
- $t \perp s$, if they are connected by antonymy relation or they are the siblings in the part of hierarchy.

2.4.3 Graph Matching

Graph matching usually is based on is-a or part-of hierarchy of the relation in the graph. Many researchers have conducted studies on this field. Do et al. [13] , calculate the similarity between internal nodes in a graph based on the similarity of children nodes. In other words, similarity between two classes is evaluated by using the similarity of child node, not terminal node.

Melnik et al. [30] presented a generic graph matching algorithm *Similarity Flooding(SF)* that uses fixed point computation to determine corresponding nodes between two graphs. The principle of the algorithm is that the similarity between two nodes must depend on the similarity between their neighbor nodes. The algorithm proceeds as follows:

1. First, given two data structures S1 and S2, translating them into graphs G1 and G2.
2. Then, using matching technique such as string-based matcher to do initial mapping between G1 and G2.
3. Based on the assumption that whenever any two nodes in models G1 and G2 are found to be similar, the similarity of their adjacent elements increases, and hence this step begins a number of iterations that each computes the similarity in each node between G1 and G2 until a fixpoint has been reached, i.e. the similarities of all model nodes stabilize.
4. Finally, depending on a variety of selecting mapping strategies. Filters select the best mappings which are then manually reviewed.

SF is flexible and extensible because it is a generic approach. However, the effectiveness and fitness for mapping different data structures such as database and ontology is unknown.

In the previous approaches, they compute coefficients between labels in the range [0,1] instead of computing semantic relations such as subsumption or equivalent relation. Hence, Giunchiglia and Yatskevich [16] proposed a method *S-Match* that computes the semantic relations(not the similarity) between nodes based on that mappings between the concepts (but not labels) assigned to nodes should be calculated. With the help of WordNet, this method proceeds as follows:

1. By using WordNet, all concepts denoted by all labels in two graphs are computed.

2. For all nodes in the two trees, compute concepts at nodes. In this step, concept is denoted by the conjunction of the concept path from root to node.

3. Using matchers in S-Match system to determine the the semantic relations between labels.

4. Using structure level matcher in S-Match system and the result in step 3 to determine the the semantic relations between nodes.

2.4.4 Matching Systems

Many matching systems have emerged during the last decade. For example, DELTA (Data Element Tool-based Analysis) [8] is a system that semi-automatically discovers attribute correspondences among database schemas by using textual similarities between data element definitions. Cupid uses linguistic and structural schema matching techniques, and computing similarity coefficients with the help of external specific thesauri. The matching algorithm in Cupid consists of three phases. The first phase computes linguistic similarity coefficients between schema element names based on some matchers such as common prefix, suffix tests. The second phase computes structural similarity coefficients by measuring the similarity between contexts. The third phase aggregates the results of the linguistic and structural matching through a weighted sum and generates a final alignment. In next paragraphs, we introduce other two matching systems *GLUE* and *COMA*.

Doan and colleagues [14] developed a system, GLUE, which uses machine learning techniques to find mappings between two ontologies. For each concept in one ontology, GLUE finds the most similar concept in the other ontology. Comparing with other machine learning approaches where only use a single similarity measure, GLUE using probabilistic definitions of several practical similarity measures. Furthermore, GLUE uses multiple learning strategies, each of which exploits a different type of information either in the data instances or in the taxonomic structure of the ontologies.

GLUE is heavily rely on instances that use to compute the joint probability distribution, $P(A, B), P(A, \overline{B}), P(\overline{A}, B)$, and $P(\overline{A}, \overline{B})$, where the term $P(A, \overline{B})$ is the notion of the probability that an instance belongs to concept A but does not belong to concept B. The joint probability distribution of the concepts is used in the similarity measures. Thereby, the application can use the joint distribution to compute any suitable similarity measure. Based on the computing of joint probability distribution and not to estimate specific similarity values directly, GLUE has the advantage that can co-work with various similarity functions which have found appropriate probabilistic interpretations.

GLUE has three kind of learner, content learner, name learner and meta-learner. The content learner uses a text classification method, called Naive Bayes learning. The name learner is similar to the content learner but uses the full name of the instance instead of its content. The meta-learner that combines the predictions of the two learners. The architecture of GLUE is composed of three main modules (Figure 2.3 [14]):

- *Distribution Estimator*: Once two ontologies has taken into the *Distribution Estimator* with their data instances, the Distribution Estimator begin to apply machine techniques to compute the joint probability distribution for every pair of concepts.

- *Similarity Estimator*: *Similarity Estimator* uses the result(probability values) from *Distribution Estimator* to compute the similarity values for probability values each pair of concepts. The output is in the form of *similarity matrix*, which will feed to the *Relaxation Labeler*.

- *Relaxation Labeler*: The *Relaxation Labeler* contains domain-specific constraints and heuristic knowledge that can support to the discovery of the most appropriate mapping. By taking into the domain-specific constraints and heuristic knowledge,

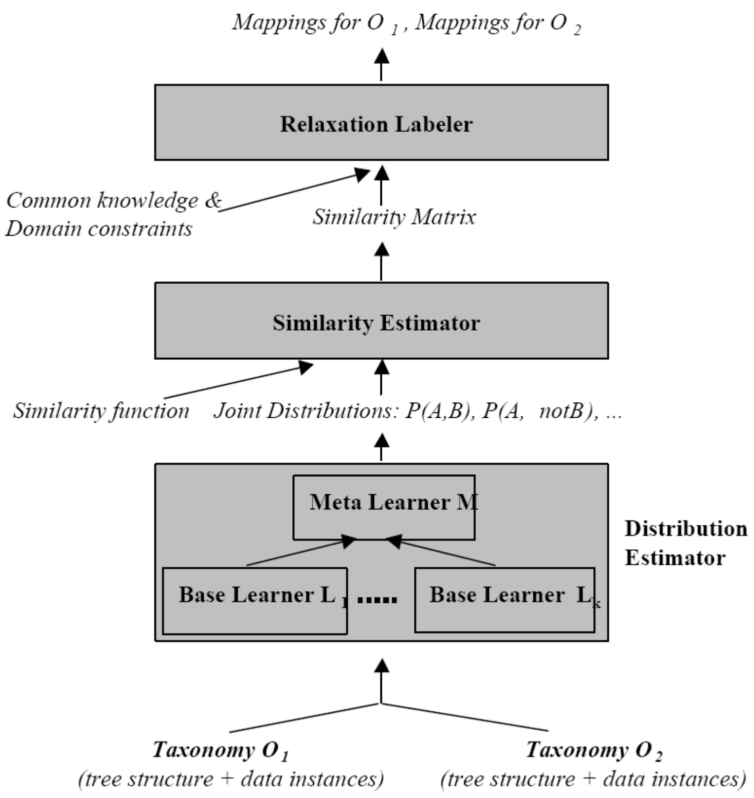

Mappings for O_1, Mappings for O_2
Relaxation Labeler
Common knowledge & Domain constraints
Similarity Matrix
Similarity Estimator
Similarity function
Joint Distributions: P(A,B), P(A, notB), ...
Meta Learner M
Base Learner L_1
Base Learner L_k
Distribution Estimator
Taxonomy O_1 (tree structure + data instances)
Taxonomy O_2 (tree structure + data instances)

Figure 2.3: The GLUE Architecture

the *Relaxation Labeler* can find the mapping which not only has high similarity, but also satisfies the domain constraints and the common knowledge. This mapping is the output of GLUE.

Do and Rahm designed a schema matching system, *COMA* [13]. Based on the idea that achieves high match accuracy for a large variety of schemas, a single technique is unlikely to be successful, COMA uses a composite match approaches more comprehensively and effectively. Moreover, COMA is a generic match system supporting different applications and multiple schema types such as XML and relational schemas. In 2005, a extended system COMA++ [4] also supports mapping between database schemas and ontologies.

The match processing in COMA is composed of numbers of iterations or one iteration. Each iteration consists of three phases(Figure 2.4 [13]):

- After two data structures are converted to the graphs, user uses Feedback matcher to capture match and mismatch information including corrected match results from

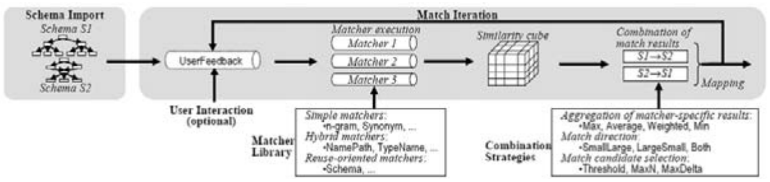


Figure 2.4: Match processing in COMA

the previous match iteration.

- In this step, multiple independent matchers chosen from the matcher library are executed. The result with k matchers, m S1 elements and n S2 elements is a k x m x n cube of similarity values

- Finally, matcher results are combined and stored in the similarity cube by the aggregation of matcher-specific results and selection of match candidates

Currently the matchers in COMA are classified into three parts: simple, hybrid and reuse-oriented matchers. Simple matchers consist of string-based matching techniques, and hybrid matchers combine of simple matchers and other hybrid matchers to obtain more accurate similarity values. Reuse-oriented matcher is a new matcher created by COMA, and it is based on the fact that many schemas to be matched are similar to previously matched schemas. The operation of reuse-oriented matcher proceeds as follows: Given two match results, match1: $S1 \leftrightarrow S2$ and match2: $S2 \leftrightarrow S3$ sharing schema S2, using the MatchCompose operation which derives a new match result, match: $S1 \leftrightarrow S3$. Note that MatchCompose operation computes the similarity measure by using *Average* strategy that is: Given n as the similarity between a and b, and m as the similarity between b and c, the similarity between a and c will be $\frac{n+m}{2}$ using MatchCompose operation.

Chapter 3

Preliminaries

Since our work aims at mapping between relational databases and ontologies, in this chapter, we first introduce preliminary studies of relational databases and OWL.

3.1 Relational Database

A database is a collection of related data or relation grouped together as a single file. The related form of these relations is depends on what the database model is applied. These databases models are primarily composed of as follows:

- Hierarchical Model: Relations are related in a parent/child tree, with each child relation having at most one parent relation
- Network Model: Similar to the hierarchical model, but each relation can have more than one parent relation
- Relational Model: Relations are related to each other by sharing a common attribute. In next section, we have the details of Relational Model.

In the past, storing data in database systems always meant to adapt the data to either the hierarchical or the network based models as mentioned above. Until the emergence of Edgar F. Codd, he devoted to the study of relational model [12] resulted in revolutionizing data storage and access fundamentally. Currently, relational database have become the most popular way to store data for nearly any of applications especially for the Web.

3.1.1 Relational Data Model

In this section we give a short introduction to the relational data model. The introduction is based on article “A Relational Model of Data for Large Shared Data Banks.” [12]

The relational data model regards a database as a set of relations or tables. In turn, each relation R consists of a collection of attributes $A_1, A_2, ...A_n$, also stated as columns. Consequently, a relation R of degree n is a subset of the Cartesian product $A_1 \times A_2 \times ... \times A_n$ with the following properties:

- Each row is a distinct n-tuple element of R.
- Each value in a tuple is atomic, i.e. it is neither composite nor multivalued.
- The order of attributes and rows are irrelevant.
- Both, relations and columns are labeled by names.
- Each attribute has a data type. There are many standard types, and each DBMS can also have their own specific types.
- Each attribute can contain the possible values .
- A attribute is said to contain a null value when it contains nothing at all.
- One or more columns comprise primary key whose values uniquely identify a row in a table.
- One or more columns comprise candidate key whose values could be used to uniquely identify a row in a table. The Primary Key is chosen among a table’s Candidate Keys.
- A relationship between two tables is created by creating a common attribute to the two tables. The common attribute must be a primary key to one table.
- A virtual table are called view made up of a subset of the actual tables.
- A one-to-one (1:1) relationship occurs where, for each instance of table A, only one instance of table B exists, and vice-versa.

- A one-to-many (1:m) relationship is where, for each instance of table A, many instances of the table B exist, but for each instance of table B, only once instance of table A exists.
- A many to many (m:n) relationship occurs where, for each instance of table A, there are many instances of table B, and for each instance of table B, there are many instances of the table A.
- Each data in relational database must not violate data integrity which describes the accuracy, validity and consistency of data.

Furthermore, when creating a database, there are some constraints which are rules or regulations imposed on data to ensure their correctness. Some important constraints are as follows:

- Unique Constraints: A unique constraint is a rule to ensure no duplicate values are in specific columns.
- Primary Key Constraints: A primary key constraint is a rule that ensure the value in primary key of columns are unique and cannot be null.
- Foreign Key Constraints: A foreign key constraint also called `referential integrity` specifies that the data in columns of a relation referred by the data in foreign key columns cannot be null.
- Check Constraints: Check constraints is a rule to ensure the validity of data in a database and to provide data integrity.

Some relationships in the relational data model are important. We now introduce the most fundamental relationships in relational databases.

- Functional dependency (FD) : Given two sets of attributes X and Y in a relation R, the FD: $X \rightarrow Y$ holds for relation R if and only if each X value is associated with at most one Y value.
- Inclusion dependency (IND) : Given a set of attributes X in a relation and a set of attributes Y in another relation, $X \subseteq Y$ holds if X is a subset of Y

- Equivalence: Given a set of attributes X in a relation R_i and a set of attributes Y in a relation R_j, X=Y holds if there exists $R_i \subseteq R_j$ and $R_j \subseteq R_i$

3.1.2 Relational Database Normalization

In creating a database, normalization is the process of organizing the database compliant with a Normal Form. Basically, it can be seen as a way to organize data in database better. There are several levels of the Normal Form, and each level requires that the previous level be satisfied. Our prototype system takes the relational database schemas as inputs, and these schemas are at least in third Normal Form (3NF). The features of 1NF to 3NF are as follows:

First Normal Form (1NF)

The goal of First Normal Form is to make sure that there is none duplicate values in the relation. If a relation is in First Normal Form (1NF), then we can conclude that each attribute of the relation is atomic.

Second Normal Form (2NF)

If a relation is in the Second Normal Form (2NF), it must be in the 1NF, too. The goal of 2NF is to remove *partial dependencies*. The partial dependency means that when the primary key are combined with several attributes, and there is some other attribute depends on just part of the primary key attributes. A relation can be said as in 2NF if it fulfills 1NF and all of the attributes which are not part of primary keys are related to the whole primary key(s).

Third Normal Form (3NF)

The goal of Third Normal Form (3NF) is to remove *transitive dependencies*. A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there is a set of attributes Z that is not a subset of any key of R, and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. A relation can be said as in 3NF if it fulfills 2NF and all attributes that are not dependent upon the primary key must be eliminated. In most situations, table (relations) in the 3NF are already fit the daily application. Much of the researches of

Relational Database Reverse-engineering and Ontology Extracting also suggests that the used relational databases must at least fulfill the 3NF.

3.1.3 Entity-Relationship model

The core foundations of relational databases are mainly in the relational data model as mentioned above. Later, the entity-relationship(ER) model [Che76] was proposed for simple yet powerful modelling technique of relational database. The ER model describes relational database as entities, relationships and attributes. An entity is a object in the real world with an independent existence, for example, a traveller, a hotel or an airline. Each entity has attributes - the particular properties that describe it. For example, an traveller entity may be described by the traveller's name, trip-location and address. An entity type defines a collection of entities that have the same attributes. The relationship type is a concept to define associations between two or more entity types, and is formally defined as a subset of the cartesian product of the participating entity types.

ER model is important for mapping between relational databases and OWL ontologies because the designers of databases often use it to model a relational database in real world. Therefore, if only reversing of engineering from relational databases to its original entity-relationship model we can capture the more semantic from the relational databases.

3.2 OWL: Web Ontology Language

Since the emerging of Semantic Web, some organizers has developed the related specification of describing Web resources and one of them is RDF. RDF was developed by W3C (World Wide Web Consortium) as a language to describe Web resources. W3C also built the RDF Schema language as an extension to RDF. The combination of RDF and RDF Schema is known as RDF(S). RDF statement are expressed in the form of triples: subject, predicate, and object. The main drawback of RDF is its expressiveness, which are basically limited to simple object, class, or property relationships. Thus, the W3C developed a new language OWL to realize the requirement for a modeling language going beyond the basic semantics of RDF Schema.

The OWL, which builds up on the RDF layer of the Semantic Web, is the W3C standard for representing ontologies on the Semantic Web. Furthermore, OWL are compatible with the Extensible Markup Language (XML) and its most immediate predecessor, namely DAML+OIL, which was deeply influenced by Description Logics. Nowadays, OWL are play a key role in Semantic Web by defining structured Web-based ontologies which allow a richer integration and interoperability of data among communities and domains.

Different from RDF language, OWL has more powerful expressiveness than RDF, e.g. additional vocabulary such as a disjointness relation between classes, transitivity and cardinality of properties, or the creation of complex classes. Depending on the expressiveness, the W3C has split OWL into three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full. The following sections to the three sublanguages is based on the standard [29] .

3.2.1 OWL Lite

OWL Lite is the most basic of the three OWL sublanguages, supporting classification hierarchies and simple constraints. Additionally, OWL Lite provides the possibility to build up subclass hierarchies and make properties optional, i.e. to impose a cardinality constraint of 0 and 1. The following are the definitions of tags for OWL Lite according to [29] .

OWL Lite RDF Schema Features

- ***Class***: A class defines a group of individuals because they share some properties. For instance, classes can be organized in a specialization hierarchy using subClassOf. Besides *Thing* is a built-in most general class.

- ***rdfs:subClassOf***: A class is a subclass of another class, which can be described by making one or more statements.

- ***rdf:Property***: A property is used to state relationships between individuals.

- ***rdfs:subPropertyOf***: Similar to subClassof, property hierarchies can be created by making one or more statements that a property is a subproperty of one or more other properties.

- ***rdfs:domain***: A domain of a property states the individuals to which the property can be applied.
- ***rdfs:range***: A range of a property states the individuals that the property may have as its value.
- ***Individual***: Individuals are instances of classes, and properties may be used to relate one individual to another

OWL Lite Equality and Inequality

OWL includes features that are related to equality or inequality.

- ***equivalentClass*** : Two classes may be stated to be equivalent. Equivalent classes have the same instances and can be used to create synonymous classes.
- ***equivalentProperty***: Two properties may be stated to be equivalent. Equivalent properties relate one individual to the same set of other individuals and may be used to create synonymous properties
- ***sameAs***: A number of different names, which refer to the same individual, can be stated to be the same by using sameAs
- ***differentFrom***: By using differentFrom, individual can be stated to be different from other individuals.
- ***AllDifferent***: A number of individuals may be stated to be mutually distinct in one AllDifferent statement.

OWL Lite Property Characteristics

There are some special identifiers in OWL Lite that are used to provide information concerning properties and their values.

- ***inverseOf***: One property may be stated to be the inverse of another property. If the property P1 is stated to be the inverse of the property P2, then if X is related to Y by the P2 property, then Y is related to X by the P1 property.

- ***TransitiveProperty***: Properties may be stated to be transitive. If a property is transitive, then if the pair (x,y) is an instance of the transitive property P, and the pair (y,z) is an instance of P, then the pair (x,z) is also an instance of P.

- ***SymmetricProperty***: Properties may be stated to be symmetric. If a property is symmetric, then if the pair (x,y) is an instance of the symmetric property P, then the pair (y,x) is also an instance of P.

- ***FunctionalProperty*** : Properties may be stated to have a unique value. If a property is a FunctionalProperty, then it has no more than one value for each individual (it may have no values for an individual). This characteristic has been referred to as having a unique property. FunctionalProperty is shorthand for stating that the property's minimum cardinality is zero and its maximum cardinality is 1.

- ***InverseFunctionalProperty***: Properties may be stated to be inverse functional. If a property is inverse functional then the inverse of the property is functional. Thus the inverse of the property has at most one value for each individual. This characteristic has also been referred to as an unambiguous property.

OWL Lite Property Restrictions

How properties can be used by instances of a class can be stated by property restrictions. There are two restrictions limit which values can be used.

- ***allValuesFrom***: The restriction allValuesFrom is stated on a property with respect to a class. It means that this property on this particular class has a local range restriction associated with it. Thus if an instance of the class is related by the property to a second individual, then the second individual can be inferred to be an instance of the local range restriction class.

- ***someValuesFrom***: The restriction someValuesFrom is stated on a property with respect to a class. A particular class may have a restriction on a property that at least one value for that property is of a certain type.

OWL Lite Restricted Cardinality

OWL Lite includes only a limited form of cardinality restrictions because they only allow statements concerning cardinalities of value 0 or 1 (they do not allow arbitrary values for cardinality, as is the case in OWL DL and OWL Full).

- ***minCardinality***: Cardinality is stated on a property with respect to a particular class. If a minCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at least one individual by that property. This restriction is another way of saying that the property is required to have a value for all instances of the class.

- ***maxCardinality***: Similar to minCardinality, if a maxCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at most one individual by that property. A maxCardinality 1 restriction is sometimes called a functional or unique property.

- cardinality: Cardinality is provided as a convenience when it is useful to state that a property on a class has both minCardinality 0 and maxCardinality 0 or both minCardinality 1 and maxCardinality 1.

3.2.2 OWL DL and OWL Full

OWL DL add additional features to OWL Lite, corresponding to the expressiveness provided by description logics . Unlike OWL Lite, OWL DL supports all language constructs of OWL, but still imposes some restrictions, e.g. an instance of a class cannot be at the same time itself a class or a property. These restrictions are required to guarantee the computational completeness and decidability of OWL DL. On the other hand, OWL Full supports the complete functionality of OWL and allows an arbitrary combination with any constructs of RDF or RDF Schema. As a result, the usage of OWL Full may result in undecidable problems. Now we begin to introduce tags of OWL DL and OWL Full according to [29] .

- ***oneOf***: (enumerated classes): Classes can be described by enumeration of the individuals that make up the class. The members of the class are exactly the set of enumerated individuals; no more, no less.

- ***hasValue***: (property values): A property can be required to have a certain individual as a value (also sometimes referred to as property values).
- ***disjointWith***: Classes may be stated to be disjoint from each other.
- ***unionOf, complementOf, intersectionOf***: (Boolean combinations): OWL DL and OWL Full allow arbitrary Boolean combinations of classes and restrictions: unionOf, complementOf, and intersectionOf.
- ***minCardinality, maxCardinality, cardinality(full cardinality)***: While in OWL Lite, cardinalities are restricted to at least, at most or exactly 1 or 0, full OWL allows cardinality statements for arbitrary non-negative integers.
- ***complex classes***:

3.2.3 The Mapping Between OWL DL And Description Logic

Description Logic (DL) is a family of logic-based Knowledge Representation (KR) formalisms. By using a structured and unambiguous way, the knowledge on an application domain can be represented and reasoned by DL. For this purpose, DL provides each of its constructs with a precise logical meaning. The basic building blocks for structuring the domain knowledge are a set of atomic concepts (unary predicates) and atomic roles (binary relations). Besides, DL provides a set of operators, called constructors, which allow to form complex concepts and roles from atomic ones.

On the other hand, since OWL DL includes all the OWL language constructs with some restrictions such as a class cannot be treated as an individual or a property, it can be processed by DL-based reasoner. As a result, OWL-DL corresponds to the Description Logic SHOIN(D). Hence, comparing with OWL Full, OWL DL is decidable, and this why we takes OWL DL as one of inputs in our prototype system. The table 3.1 is the mapping between OWL and basic DL notations.

Table 3.1: OWL Constructors and Corresponding Description Logic Syntax

Constructor	Description Logic Syntax	Example
`intersectionOf`	$C_1 \sqcap ... \sqcap C_n$	Location $\sqcap$ Taipei
`unionOf`	$C_1 \sqcup ... \sqcup C_n$	DaanChiu $\sqcup$ ShinyiChiu
`complementOf`	$\neg C$	$\neg$TaipeiCity
`oneOf`	$\{x_1 ... x_n\}$	{Red, Blue, Yellow}
`allValuesFrom`	$\forall P.C$	$\forall$locatedIn.TaipeiCity
`someValuesFrom`	$\exists P.C$	$\exists$isPassedBy.BusNo_1
`hasValue`	$\exists P.x$	$\exists$ locatedIn.{ind_TaipeiCity}
`minCardinality`	$\geq nP$	$\geq$ 3hasTripElement
`maxCardinality`	$\leq nP$	$\leq$ 2hasTripElement
`cardinality`	$= nP$	= 1hasTripElement

3.3 Differences Between Database Schema and Ontology

When mapping relational databases to OWl ontologies using relational database schemas, differences between them should be first taken into account. Now we introduce these differences based on [33] .

- In general, the aim is to preserve the integrity of data itself when creating database schemas. However, for ontologies, data themselves can be seen as ontologies. Moreover, in many cases, an ontology will not have any instance data at all, and a result of an ontology query can include elements of the ontology itself but a result of an database query is data or refer to other data.

- Comparing with the ontology, database schema itself provide very little semantic, whereas ontology is logic-based, and hence the semantics is explicitly represented.

- Database schemata are not sharable or reusable, usually they are part of an integrated system and is rarely used apart from it, whereas ontologies are by nature reusable and typically extent other ontologies, and they are not bound to a specific system.

- In general, database schema development and maintenance is a centralised process, whereas ontology development is more de-centralised.
- Database schema often takes into account effects of each change operation on the data. However, in ontologies, the number of knowledge representation primitives is much higher and more complex
- Databases make a clear distinction between schema and instance data, whereas classes and instances can be the same in some knowledge representation languages used for ontology modelling such as RDFS.

Chapter 4

Mapping Approach

In this chapter, we begin to introduce our approach to mapping a relational database to OWL ontologies. As mentioned in Chapter 2, the approaches in the related field are mainly composed of two parts. One is “to extract ontologies from a relational database” and the other is “directly to map a relational database to existing OWL ontologies.” The former primarily focus on finding rules about how to extract classes and properties of OWL ontologies from tables and columns of a relational database by analysing keys, data and so on. The latter primarily discovers a new method of mapping a relational database to domain-related OWL ontologies and this is our object. Besides, a fully-automatic mapping approach in this field is still in the very early stage to be implemented since the related technique such as the natural language processing (NLP) has not been yet mature enough to deal with all mapping problems. Therefore, the use of a semi-automatic mapping approach is seen as the practical short terms solution.

We proposes a semi-automatic approach that directly to map a relational database to OWL ontologies. As far as we know, the methods of directly to map a relational database to existing OWL ontologies are less related to the ones of extracting OWL ontologies from a relational database. Besides although some approaches using a two-phases mapping method, which first extracts OWL ontologies from a relational database and then maps this extracted OWL ontologies to existing OWL ontologies, it would have the drawback of loosing more information, that is the extracted OWL ontologies from a relational database would lose some semantic information, and the mapping process between OWL ontologies would also lose some information. However, our approach immunizes this problem even though using the contributions of extracting OWL ontologies from a relational database.

4.1 Overview of the Approach

Our approach of directly mapping a relational database to OWL ontologies is based on the concept of cluster analysis in data mining. There are many data mining techniques and one of them is cluster analysis [35] whose purpose is to find groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups. Extending this purpose to our approach is to find a matching classes group (MCG) for every table such that every found class of MCG of the table has the same features and the final mapping result of the table must be in its MCG and satisfy the mapping consistency. Mapping consistency states that the mapping results should not violate the fact expressed in the relational database.

Since the final mapping results must be in the MCGs of tables and the most important features for classes of OWL ontologies are their properties, especially for object properties, our approach of first phase is to map foreign keys to object properties to get the features of MCGs of the related tables and the second phase finds MCGs of tables using the results of the first phase as inputs. After finishing these phases, some new properties or classes can be created based on the mapping rules. Furthermore, there are two assumptions in this approach.

- Assumption one: The mapping results of the first phase are accurate, that is, these mapping results are consistent with the fact expressed in the real world.
  - Rationality: Our approach is semi-automatic and hence users will need to refine the final mapping results after finishing the phase one. For this purpose, we expected these mapping results refined by users are accurate.
- Assumption two: When using our approach to mapping a relational database to OWL ontologies, the domain size described by OWL ontologies should subsume the one described by the relational database.
  - Rationality: Relational databases are less semantic model comparing to OWL ontologies. Besides, the domain size of OWL ontologies usually increase gradually with time but relational databases do not. Hence for some relational

databases whose domain size are not subsumed by OWL ontologies currently, OWL ontologies will subsume these domain in the long run.

Now we first briefly overview the approach implemented by the system in Figure 4.1 and detail our approach in the following sections.

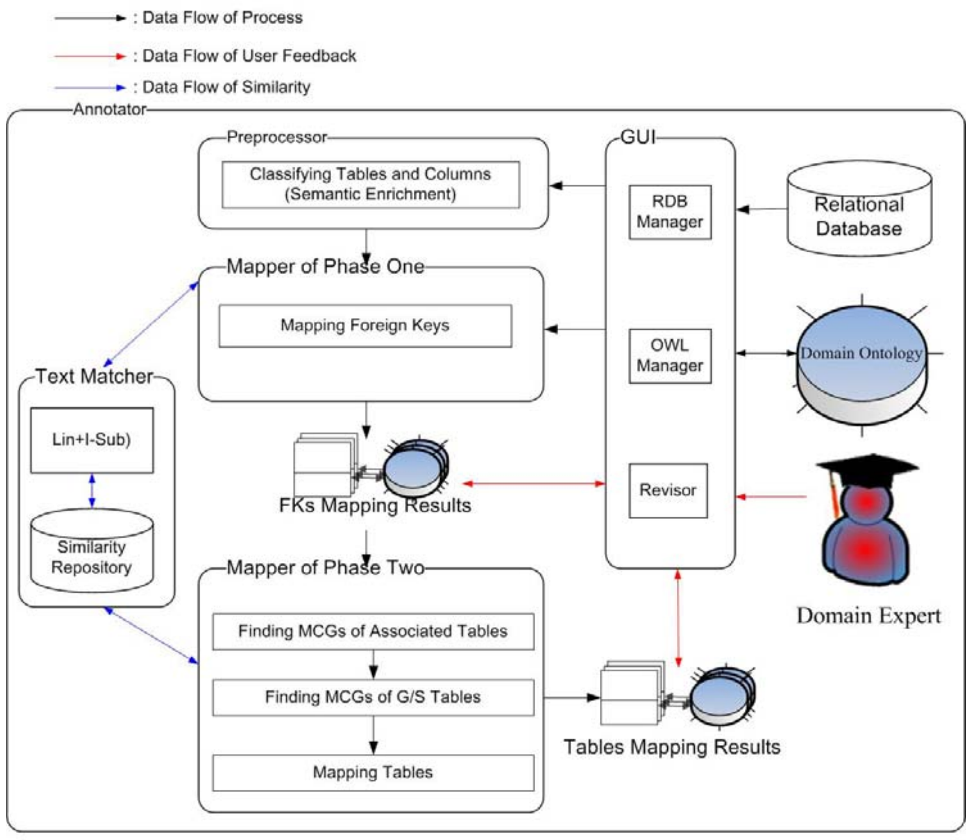


Figure 4.1: System Architecture

- **Preprocessing**: Since mapping between a relational database and OWL ontologies is usually from a less semantic model to a more semantic one, we will engage in preprocessing the relational database schema by the help of semantic enrichment to get more semantic data from the relational database. The semantic enrichment can be done by classifying tables and columns of a relational database as in Table 4.1.

- **Phase One**: After classifying tables and columns, we will get the related semantic information for every column and we will use this information to map foreign keys of a relational database to object properties of OWL ontologies.

Table 4.1: Classification of Tables and Columns

Classification of Tables		
Independent Tables	Associated Tables	Generalization/Specialization(G/S) Tables

Classification of Columns					
Foreign Keys			Non Foreign Keys		
Base FKs	Part-Of FKs	IS-A FKs	Numeric Group	String Group	Time Group

- **Phase Two**: The object of the second phase is to find MCGs of tables then map tables to classes. According to the structure of tables, we will first to find MCGs of associated tables whose inputs are the mapping results in phase one and hence the assumption one is taken into this mapping process. Then the MCGs of some cases of G/S table can be found by using the MCGs of associated tables. Besides, the MCGs of other cases of G/S tables and independent tables will all be regarded as the collection of all classes of OWL ontologies. Finally, after system completes finding the MCG for every table, the mapping between tables and their MCGs will begin to be dealt with.

4.2 Preliminary Definitions

We first define some terms of the relational database schema and OWL ontologies used in our approach.

Definition 4.2.1 (Relational Database Schema)**.** Let $T = t_1, t_2, ..., t_n$ be the set of tables from a relational database schema $RDBS$, where

- Each table t_i has a set of columns $COL(t_i) = c_1, c_2, ..., c_i$
- Each table t_i has a set of primary keys $PK(t_i)$
- Each table t_i has a set of foreign keys $FK(t_i)$

- The foreign key $fkey$ refers to the table $Ref(fkey)$
- The table $RefBy(fkey)$ referred by the foreign key $fkey$
- The tables $SubT(t_i)$ are the set of sub-table of t_i
- The table $SupT(t_i)$ is the super-table of t_i

Definition 4.2.2 (OWL Ontologies)**.** Let $T = cls_1, cls_2, ..., cls_n$ be the set of classes from OWL ontologies, where

- Each class cls_i has a set of object properties $OP(cls_i)$
- Each class cls_i has a set of datatype properties $DP(cls_i)$
- Each class cls_i has a set of possible subclasses $SubC(cls_i)$
- Each class cls_i has a set of possible superclasses $SupC(cls_i)$
- Each object property op_i has a set of all possible domain classes $DOM(op_i)$
- Each object property op_i has a set of all possible range classes $Range(op_i)$

4.3 Classifying Tables and Columns

Our goal in first step is to classify tables and columns into groups as Table 4.1. Classifying these entities is necessary to reach the more accurate mapping results when applying the following intuitions.

- Intuitively, two entities are hard to be mapped if their external structures are different and therefore we group tables of the relational database schema into three kinds of table: Independent Tables, Associated Tables, and G/S Table. We map foreign keys to object properties and non-foreign keys to object or datatype properties. Because object properties in OWL can link individuals of domain classes to individuals of range classes and foreign keys in the relational database are their counterparts since for foreign keys, they can link the instances of table they store in to the instances of table they refer to. But as for non-foreign keys, their counterparts in OWL ontologies are datatype properties because both of them can not link two individuals or instances to each other. Nevertheless, non-foreign keys can also be mapped to object properties because there are some columns in the relational database that can independently represent entities, i.e., these columns can be the range classes of object properties. Furthermore, foreign keys and non-foreign keys can be further classified into various groups and this will discussed in the following sections.

- Intuitively, two entities have different internal structure, especially for the different datatype, will be difficult to match each other and hence when mapping between non-foreign keys and datatype properties, the datatype between them will be taken into consideration.

Since the useful semantic data is not explicitly defined in the relational database schema, in other words, the information such as which table belongs to which group is invisible and hence we will first engage in the semantic enrichment of the relational database to extract some implicit information which can be used to find some classifications of tables and columns such as G/S tables. Therefore, semantic enrichment is the task of gathering additional semantic information which is not explicitly available from the relational database system [6]. In the research of transforming the relational database schemas to other data models such as object-oriented databases schemas, semantic enrichment is considered an important process and hence many researches in this area had made some helpful contributions. In general, these results of research are presented as rules. Furthermore, in the topic of extracting OWL ontologies from the relational database, the processes of extracting OWL ontologies also primarily focus on finding rules through analyzing the semantic of the relational database schema. Therefore, we will greatly take advantage of the contributions in these field to classify tables and columns to get their semantic.

4.3.1 Classifying Tables

In relational data model, there are some tables representing the real world entities whose instances can be identified exclusively through its own attributes and have no any relationship with other tables. In this thesis, they are called independent tables since the semantic of this kind of table usually explicitly reflects in the relational database schema and the following is its definition.

Definition 4.3.1. For table t_i , t_i is an independent table if $FK(t_i) = \Phi$ and no other foreign keys of tables refers to t_i

Associated tables are the main components in the relational data model representing the various relationship between tables such as one-to-one relationship, many-to-one relationship and many-to-many relationship. Among them, a relationship table will be created in order to express the many-to-many relationship between tables. A relationship table in the relational data model denotes a set of tuples, each one is composed of instances of the entities that participate in the relationship. The number of the participation of these entities in a relationship constructs a n-arity relation. For example, for tables "package(packageID, numOfDays, departed,...)" and "spot(name, location)" in the relational database schema 5.3, package is a table used to store the related information of the journey. Therefore, a package can have many spots and a spot can be included by many packages. In order to represent many-to-many relationship between "package" and "spot", a relationship table "spotDetail(packageID, sname)" , which combining primary keys "packageID" and "sname" as foreign keys, need to be created. The followings are the definitions of associated tables and relationship tables.

Definition 4.3.2. For table t_i , t_i is an associated table if there exist a foreign key c_i which is not an IS-A FK or some other foreign keys of tables refer to t_i

Definition 4.3.3 (Relationship Table)**.** For tables t_1, t_2,..., t_i and t_j , t_j is a relationship table if $PK(t_j) = FK(t_j) = PK(t_1) \sqcup PK(t_2) \sqcup \ldots \sqcup PK(t_1)$

To represent inclusions between the sets of instances of two entities, so called Generalization/Specialization(G/S) tables are used. G/S tables states the inheritance of properties from a more general entity to a more specific one. In other words, two tables have a G/S relationship if every instance of one table is also an instance of the other table. In general, many studies [24] had considered that finding the tables have inheritance relationships is equal to finding the tables have the subsumed relationship between their own primary keys and hence the definition for G/S tables is as follow.

Definition 4.3.4 (G/S table)**.** For tables t_i, t_j , t_i and t_j are G/S tables if $PK(t_i) \subseteq PK(t_j)$ and exist $Ref(c_i) = t_j$, where $c_i \in FK(t_i)$

For example, for tables "accom(<u>hotelCode</u>, location)" and "resortHotel"(<u>hotelCode</u>, resort) in our example of relational database schema 5.3, where hotelCode of resortHotel refers to hotelCode of accom. These two tables are G/S tables since there exist $PK(resortHotel) \subseteq PK(accom)$ and $Ref(hotelcode) = accom$.

By using the definitions, independent tables and associated tables can be automatically detected by the system, but as for some G/S tables user interaction is required because the relational data model is known to be a semantically poor model and the concept of inheritance relationship cannot be directly represented in the relational database. Besides, in the designing of the relational database schema, designers rarely consider the inheritance relationships between tables because the main goal of designing a relational database is to create the binary relationship between two tables. Therefore, as opposed to OWL ontologies, inheritance relationships are less and not explicit in the relational database schema. In order to find the tables t_i and t_j satisfying the definition of G/S table, we need to verify every instance of $PK(t_i)$ is also an instance of $PK(t_j)$. But since we have no instance in hand can proof it, some cases need querying users to find the G/S tables.

Now discussing these cases, for tables t_i, t_j and foreign key c_i , t_i and t_j can at least have "is-part-of" and "has-part" relationship [24] if $c_i \in FK(t_i)$, $c_i \in PK(t_i)$ and $Ref(c_i) = t_j$. If the condition $|PK(t_j)| = 1$ pluses to this case, we can further make sure t_i and t_j exist a G/S relationship since we know every instance of t_i is also an instance of t_j. But if the conditions $|PK(t_i)| > 1$, $|PK(t_j)| > 1$ and $|PK(t_i)| \geq |PK(t_i)|$ are satisfied, whether t_i and t_j have a G/S relationship can only be decided by interacting with users. For instance, if $PK(t_i) = \{c_a, c_b\}$, $PK(t_j) = \{c_c, c_d\}$ and c_a is also a foreign key of t_i referring to c_c of t_j, t_i and t_j can not be made sure if they have a G/S relation since whether instances of c_b are subsume by instances of c_d or not can not be proofed.

4.3.2 Classifying Foreign Keys

After the process of classifying tables, the semantic of different kind of foreign keys can also be found concurrently [24]. These kinds of foreign keys are defined as follows.

Definition 4.3.5 (Base FK). For table t_i and its foreign key c_i, c_i is a base FK if $c_i \nsubseteq PK(t_i)$

Definition 4.3.6 (IS-A FK). For super-table t_j, its sub-table t_i and foreign key c_i of t_i, c_i is an IS-A FK if $c_i \subseteq PK(t_i)$ and $c_i \subseteq PK(t_j)$

Definition 4.3.7 (Part-Of FK). For table t_i and its foreign key c_i, c_i is a Part-Of FK if c_i is not an IS-A FK and $c_i \subseteq PK(t_i)$

4.4 Matchers in the Approach

Before describing the method for mapping foreign keys to object properties we are first to discuss the matchers and source of computing the similarity using in this approach.

4.4.1 Matchers

Various matching techniques have been introduced in Chapter 2 , especially for the string-based and WordNet-based matchers. Usually mapping is a complex process and a single kind of matcher often can not satisfy the mapping problem since different situation can be happened when matching different information structures, and a matcher is frequently designed for specifying applications. For this reason, some matching systems use composite or hybrid matchers. Composite matcher is a matcher that combines the results of multiple single matchers and hybrid matcher incorporates the features of several matchers into one compound matcher. Furthermore, most of the time is spent on computing the similarities between entities when dealing with the mapping problem. Therefore, an efficiency matcher is required. In this approach, we use a hybrid matcher combining Lin and I-Sub measures. These two measures are complement based on the fact that Lin can be used as a WordNet-bases matcher and I-Sub can be used as a string-based matcher and besides, the designing concept of I-Sub comes from the intuitions of Lin. Now explaining this hybrid matcher.

In the field of the Information retrieval(IR), the following formula can be represented the information content of some word w in a document.

$$IC(w) = log(\frac{1}{P(w)})$$

Intuitively, two concept are similar if they share some mutual information content and hence Resnik[34] uses the following equation as a method to evaluate the similarity between two concepts.

Definition 4.4.1 (Resnik Similarity)**.**

$$Sim(c1, c2) = \max_{c \in Sup(c1,c2)} [-\log P(c)]$$

, where $P(c)$ is the probability of encountering an instance of concept c and $Sup(c1, c2)$ is a set of superconcept of c_1 and c_2

If all words in a WordNet tree are regarded as a document, the similarity of two words can be determined by their common information contents. Besides, since a word in WordNet tree can be seen as a concept, any similarity of two words can depend on their common information contents in WordNet tree.

Since the equation of Resnik only considers the commonality between two concepts, Lin [25] proposed a similarity measure normalizing Resnik's equation based on the three intuitions. In his thesis, intuitions are described as follow.

- Intuition 1: The similarity between A and B is related to their commonality. The more commonality they share, the more similar they are.
- Intuition 2: The similarity between A and B is related to the differences between them. The more differences they have, the less similar they are.
- Intuition 3: The maximum similarity between A and B is reached when A and B are identical, no matter how much commonality they share.

Commonality or difference between two concepts can be quantified by using the information contents of these two concepts and hence Lin extended Resnik's equation to the following formula:

$Sim_{lin}(c_1, c_2) = \dfrac{2 * IC(LCS)}{IC(c_1) + IC(c_2)}$, where LCS is the least common subsumer of c_1 and c_2.

Stoilos et al. [36] proposed a string-based matcher called I-Sub. The main idea of I-Sub comes from Lin measure, that is, the similarity among two entities is related to their commonalities as well as to their differences and hence the following equation is proposed.

$$Sim(s_1, s_2) = Comm(s_1, s_2) - Diff(s_1, s_2) + Winkler(s_1, s_2)$$

,where

$$Comm(s_1, s_2) = \frac{2 * \sum_{i=1}^{n}(length(maxComSubString_i))}{length(s_1) + length(s_2)}$$

$$Diff(s_1, s_2) = \frac{uLen_{s_1} * uLen_{s_2}}{p + (1-p) * (uLen_{s_1} + uLen_{s_2} - uLen_{s_1} * uLen_{s_2})}$$

$$Winkler(s_1, s_2) = commonPrefixLenght * 0.1 * (1 - Comm(s_1, s_2))$$

Note that in our approach, we ignore the formula $Diff(s_1, s_2)$. Because as opposed to the names of entities in OWL ontologies, the ones of entities in relational databases are often incomplete and abbreviated. Furthermore, I-Sub is designed for mapping between OWL ontologies. Therefore, when mapping between a relational database and OWL ontologies, the fact that some entities whose similarities should be high will not be shown due to the formula $Diff(s_1, s_2)$.

Now using two strings "hasTransportation" and "traffic" as an example to describe how a hybrid matcher combining Lin and I-Sub measures computes the similarity in our approach. Before calculating the similarity of these two string, they need to be tokenized first as described below.

- Tokenization: In information retrieval, a string is considered as a set of words also called bag of words. Besides, in the area of Computer Science researchers tend to use multiple words to represent variables or real world entities. Hence, in order to get the more accurate meaning of strings, we need to tokenize these strings by segmenting them into sequences of tokens. In this approach, we recognize four tokenizing processes:

  - Replacing the punctuation character with the blank.
  - Recognizing the upper case as a beginning of a word.
  - Removing digits.
  - Removing stopwords.

After tokenizing, “hasTransportation” becomes “transportation” and “traffic” does not change. Then using our hybrid matcher to compute the similarity between these two strings. Since “transportation” and “traffic” can be found in WordNet their information contents are 9.054 and 10.170 respectively and the information content of LCS for these two strings is 5.894. Therefore $Sim_{lin}(transportation, traffic) = 0.613$. Note that all information contents are computed in advance [28]. If this example is revised to “hasTrans” and “traffic”, “hasTrans” will become “Trans”. I-Sub will compute the similarity in this case since trans can not be found in WordNet.

4.4.2 Source of Computing the Similarity

In order to increase the mapping performance, many researches make full use of the information related to the mapped entities. In general, these information can be summarized as follows.

- Local similarity: Some information uses for describing the entities is local information. For instance, we usually give a name as a meaning of an entity and this name is the local description of this entity. Hence, many approaches use matchers to compute the similarity of entities by their local descriptions.
- Internal similarity: Similar entities usually have the similar internal structures. In other words, if two entities are with similar internal structures, they will have a chance to match each other. For instance, we can compute the internal similarity by the datatypes of two entities such as “int” and “float”, then the result can influence the final similarity measure between these two entities.
- External(Relational) similarity: Similar to internal similarity, two entities will be possible to match each other if their neighbors are similar. We compute the similarity of their neighbors as the external similarity. Then, the final result of the similarity measure between two entities will be influenced by the external similarity.

4.5 Mapping Foreign Keys to Object Properties

Many researches [30] claim that the similarity of two entities will be influenced by their neighbor entities. Hence, when mapping a foreign key to some possible object properties, if only considering the information foreign key itself has, there will cause a low accuracy of mapping results. Besides, since some semantic of foreign keys can be gotten by classifying tables a foreign key can have different matching level depend on the semantic it has. Combining above mentions, two principals of mapping foreign keys to object properties are composed as follows:

- Principal 1: The similarity between two entities should be influenced by their neighbor entities.
- Principal 2: Foreign keys can have different level matching depend on the semantic they have.

Therefore, the principals and source of computing similarity are considered when mapping foreign keys to object properties. In the following sections, we introduce how to map various kinds of foreign keys to object properties except IS-A FKs. Because as for OWL ontologies, it does not use object properties to declare subclass and superclass relationship between classes but use a built-in vocabulary "owl:superClassOf" or "owl:subClassOf" and hence IS-A FKs will be ignored in the matching process.

4.5.1 Mapping Base FKs

As defined in 4.3.5, the semantic of a base FK has explicitly expressed the fact that it can relate one table to another table and as for object properties which can link individuals each other between domain classes and range classes. Therefore, for one base foreign key and one object property, their total similarity can be combined the local similarity and the external similarity to improve the performance of mapping results. The related definitions of these similarities are defined as follows.

Definition 4.5.1 (Local Similarity for Two Entities)**.** For two entities e_i and e_j, their local similarity can be expressed as $Sim_{local}(e_i, e_j)$ which is the result using matchers to compute their local information such as names.

For a foreign key c_i, the table c_i stores in and the table c_i refers to will be the main structures for c_i. The counter parts in object property op_i are its domain classes and range classes. Hence, the external similarity of c_i and op_i can combine their domain similarity $Sim_{domain}(c_i, op_i)$ and range similarity $Sim_{range}(c_i, op_i)$. These two similarities are defined as follows.

Definition 4.5.2 (External Similarity of FKs and OPs)**.** For foreign key c_i of table t_i referring to table t_j and object property op_i which has n related domain classes $\{dc_1, dc_2, \ldots, dc_n\}$ and m related range classes $\{rc_1, rc_2, \ldots, rc_n\}$, the exteranl similarity of c_i and op_i is

$$Sim_{external}(c_i, op_i) = Sim_{domain}(c_i, op_i) + Sim_{range}(c_i, op_i)$$

where

- $Sim_{domain}(c_i, op_i) = \max\limits_{1 \leq x \leq n} Sim_{loc}(t_i, dc_x)$ is the domain similarity of c_i and op_i.
- $Sim_{range}(c_i, op_i) = \max\limits_{1 \leq y \leq m} Sim_{loc}(t_j, rc_y)$ is the range similarity of c_i and op_i.

According to the principal 2 foreign keys can have different level matching depend on the semantic they have. In this approach, two matching levels, which are level one matching and level two matching, are used. Level one matching represents the matching between entities only considers the explicit information they have. As opposed to level one matching, level two matching uses implicit information to match entities. Hence, since a base FK in this approach is regarded as an entity which has no implicit information it will be matched up to level one and its similarity is defined as follows.

Definition 4.5.3 (Similarity for Base FKs and Object Properties)**.** For base foreign key c_i and object property op_i, the similarity between them can be computed as:

$$Sim_{one}(c_i, op_i) = Sim_{local}(c_i, op_i) + Sim_{external}(c_i, op_i)$$

, where

- $Sim_{local}(c_i, op_i)$ is the local similarity.
- $Sim_{external}(c_i, op_i)$ is the external similarity.

Since for each base foreign key there are many possible object properties to match, we give the priority to the matching candidate based on the rank of their similarity. For example, the best matching candidate for base foreign key c_i can be defined as follows.

Definition 4.5.4. For base foreign key c_i and n numbers of object properties $\{op_1, op_2, \ldots, op_n\}$, the best matching candidate from these object properties for c_i is op_i if :

$$Sim(c_i, op_i) = \max_{1 \leq x \leq n} Sim_{one}(c_i, op_x)$$

4.5.2 Mapping Part-Of FKs

For any two individuals I_1 and I_2 in OWL ontologies, if I_1 is part of I_2 and I_2 has part of I_1, I_1 and I_2 will have inverse relationship and this kind of relation can be easily claimed in the OWL ontologies as two object properties which have inverse relation. But as for the relational database schema, inverse relationship between tables can not be explicitly expressed. Because for any two tables t_i and t_j, if they have inverse relationship in the real word, database designers usually regard this kind of relationship as the common binary relationship and hence may design that t_i uses a foreign key to refer to t_j and the fact that t_j can also use a foreign key to refer to t_i would not be captured.

As mentioned above, the fact that two tables have the inverse relationship will be represented incompletely, in other words, some implicit information is not expressed in the relational database schema. Hence, extracting these implicit information is necessary to improve the performance of mapping result. For this purpose, our approach takes advantage of Part-Of FKs to find the inverse relationship between tables. Because for a table t_i and its Part-Of FK c_i referring to table t_j, the fact that t_i and t_j have inverse relationship is satisfied since both of them have the common primary key describing themselves.

Mapping a Part-Of FK to object properties can adopt a level two matching since this kind of FK has implicitly expressed the information about two tables exist an inverse relationship. In level one matching for a Part-Of FK c_i, the behavior of computing similarity between c_i and object properties is the same as the one of a base FK. But when reaching the level two matching, we will create a conceived foreign key c_j to express the implicit information of c_i. The name of c_j is the same as c_i and now c_j becomes a foreign key of the table c_i refers to and c_j refers to the table c_i stores in. Hence, if combining level one and level two matching for c_i and n numbers of object properties, there will be 2n numbers of matching results for c_i. n numbers of them are the matching results using explicit information c_i to match and the others are the matching results using implicit information c_i to match. Now we define the best matching candidate of c_i.

Definition 4.5.5. For tables t_i and t_j, if t_i has a Part-Of foreign key c_i referring to t_j and c_i will match to n numbers of object properties $\{op_1, op_2, \ldots, op_n\}$, a conceived foreign key c_j of t_j referring to t_i will be created and the best matching candidate from these object properties for c_i is op_i if :

$$Sim(c_i, op_i) = \max((\max_{1 \le x \le n} Sim_{one}(c_i, op_x)) \sqcup (\max_{1 \le y \le n} Sim_{two}(c_i, op_y)))$$

where

- $Sim_{one}(c_i, op_x)$ is the level one matching result.
- $Sim_{two}(c_i, op_y)$ is the level two matching result.

4.5.3 Mapping FKs of Relationship Tables

When calculating the external similarity between foreign keys and object properties, we usually take the tables foreign keys store in and domain classes of object properties as input to compute the domain similarity but as for foreign keys of relationship tables we doesn't always do that. Because relationship tables are created to facilitate the "many-to-many" relationship betweens tables and not to represent entities in the real word. Furthermore, on one hand, one relationship table can represent n-arity relationship, but on the other hand, OWL ontologies can only support unary or binary relationship.

Hence for a relationship table t_i which has a set of foreign keys $\{c_1, c_2, \ldots, c_n\}$, when mapping these foreign key to object properties, two cases are considered as follows to capture the semantic of relationship table as far as possible.

- Case 1: If $n = 2$ and $COL(t_i) = FK(t_i)$, the table used to compute the domain similarity for c_1 will be the the table referred by c_2 and the table used to compute the domain similarity for c_2 will be the table referred by c_1

- Case 2: If $n \geq 2$ or $COL(t_i) \neq FK(t_i)$, every table used to compute the domain similarity for c_n will be the relationship table itself

4.6 Mapping Tables

In the second phase, we devise a new method to map tables of a relational database to classes of OWL ontologies. This new method is based on the idea of “Cluster Analysis”. We will use the cluster analysis to find the MCG for every table and every class of the found MCG will satisfy the “Mapping consistency”. For example, if two associated tables t_i and t_j express the face that t_i can relate to t_j through its foreign key c_i, the mapping result of t_i could also related to the one of t_j through the one of c_i.

Basically, the object of finding MCG for every table can be easily to be reached by the help of the related mapping results. We first take the mapping results in phase one as inputs when finding MCGs for associated tables. In phase one, the object of approach is to map foreign key to object properties and what the method we used has been discussed in the previous sections. After finishing this phase, user will first obtain the mapping results between foreign keys and object properties and beginning to refine the mapping results, i.e., to revise some best mapping results which are not correctly matched. Hence, we can assume these mapping results are accurate then the MCGs for associated tables can be found with these results. Finally, once finishing the finding of the initial MCGs of associated tables, we can find MCGs for some cases of G/S table using the MCGs of associated and the principal of mapping consistency. Now detailing this method.

4.6.1 MCGs of Associated Tables

In the phase one, if one foreign key need matching to n numbers of object properties, n numbers of similarities between them will be required to be computed. It seems as demanding a lot time to tackle this task. In fact, many methods proposed[16] [30] in matching field need an initial mapping using the element level matching. This initial mapping will usually compute all possible local similarities between attributes of entities or/and entities of two schemas to determine the initial mapping results. Moreover, in OWL ontologies, the number or the increased rate of object properties is often much less than the classes and hence the results of our experiment showed that time taken to compute the similarities in the phase one is acceptable. But since the situation of classes in OWL ontologies is different from object properties and the mapping result of this phase must satisfy the mapping consistency we should take a more efficiency way to deal with mapping between tables and classes. For this reason, when mapping one associated table to n numbers of classes, instead of computing every matching class between them, we only need to compute some necessary matching classes which lead to every mapping result of associated table satisfies the definition 4.6.1 and these necessary matching classes are the members of MCG.

Definition 4.6.1 (Mapping Consistency for Associated Tables)**.** For associated tables t_i, its MCG must satisfy that no matter which class of its MCG chosen by users as a mapping result this chosen class will be a qualified one, i.e., this mapping result is consistent with the fact expressed in t_i.

As mentioned above, the mapping results of mapping foreign keys to object properties in phase one are assumed correct and these mapping results are right the features of the classes which associated tables expected to map. Therefore, the MCG for every associated table can be easily found by the help of these results. First, we first to find all possible matching classes for every associated table based on these results. For an associated table t_i, we denote $MCG(t_i)$ as a set of all possible matching classes group for t_i. $MCG(t_i)$ can be found by intersecting domain or range classes of the mapping results of the foreign keys for t_i and range or domain classes of the mapping results of the foreign keys referring to t_i. The definition of matching classes group(MCG) is as follows.

Definition 4.6.2 (Matching Classes Group(MCG))**.** For an associated table t_i, which has n numbers of foreign keys $\{cout_1, cout_2, \ldots, cout_n\}$ and is referred by m numbers of foreign keys $\{cin_1, cin_2, \ldots, cin_m\}$, if the mapping results in phase one for $\{cout_1, cout_2, \ldots, cout_n\}$ and $\{cin_1, cin_2, \ldots, cin_m\}$ are $\{opin_1, opin_2, \ldots, opin_n\}$ and $\{opout_1, opout_2, \ldots, opout_m\}$ respectively, the matching classes group for t_i will be as follows.
$MCG(t_i) = DOMRange(opout_1) \sqcap DOMRange(opout_2) \sqcap \ldots \sqcap DOMRange(opout_n) \sqcap DOMRange(opin_1) \sqcap DOMRange(opin_2) \sqcap \ldots \sqcap DOMRange(opin_m)$
Note that :

- $DOMRange(opout_n) = Range(opout_n)$ if the mapping result of $cout_n$ is from the level two matching.
- $DOMRange(opout_n) = DOM(opout_n)$ if the mapping result of $cout_n$ is from the level one matching.
- $DOMRange(opin_m) = DOM(opin_m)$ if the mapping result of cin_n is from the level two matching
- $DOMRange(opin_m) = Range(opin_m)$ if the mapping result of cin_n is from the level one matching

For example, supposing that there exist no foreign key referring to table t_j, t_j has one foreign key cin_1 referring to t_i and t_i has two foreign keys $cout_1$ and $cout_2$, besides, object property $opin_1$ is the mapping result of cin_1 from level one matching and object properties $opout_1$ and $opout_2$ are the mapping results of $cout_1$ and $cout_2$ from level one and level two matching respectively. Hence, these mapping results tell the fact that the class t_i expected to map is the domain of $opout_1$, the range of $opout_2$ and the range of $opin_1$ and the class t_j expected to map is domain of $opin_1$. Finally, in order to satisfy the principal of mapping consistency, $MCG(t_i)$ must be the intersection of all domain classes of $opout_1$, all range classes of $opout_2$ and all range classes of $opin_1$ and in the same way $MCG(t_j)$ is equal to all domain classes of $opin_1$. In this way, choosing any one class from $MCG(t_i)$ and $MCG(t_j)$ respectively will satisfy the mapping consistency, i.e., satisfying the fact that t_j can related to t_i through cin_1 in the real word and any one class of $MCG(t_j)$ can related to any one class of $MCG(t_j)$ through $opin_1$.

4.6.2 MCGs of G/S Tables and Other Tables

The second main step in phase two is to find MCGs of G/S tables. Similar to associated tables, we also find all possible matching classes for G/S tables. Nevertheless, different from the associated tables, which can take advantage of the mapping results of foreign keys, G/S tables have no such information because their foreign keys use to refer to the supertable are kind of IS-A links which are not necessary to map to the object properties. But in some case, finding MCGs of G/S tables can still reach the same effect by using the MCGs of associated tables in first step. These cases are those tables which are the members of both associated tables and G/S tables and we will first deal with such cases.

First, we find those G/S tables which constitute a generalization/specialization tree and exist at least one table which is also a member of associated tables. Since these members have the initial MCGs in first step, these initial results will be used to find MCGs of any other members in these trees. The MCGs of G/S tables should satisfy the following mapping consistency.

Definition 4.6.3 (Mapping Consistency for G/S Tables)**.** For a subtable t_i and its supertable t_j , their matching classes of groups $MCG(t_i)$ and $MCG(t_j)$ must satisfy one of the following conditions

- It must exist at least one matching class in $MCG(t_j)$ which is the superclass of the mapping result of $MCG(t_i)$ chosen by users.
- It must exist at least one matching class in $MCG(t_j)$ which can lead to the same mapping result as the one of $MCG(t_i)$ chosen by users.

Taking a example to describe the processes of finding MCGs for G/S tables, for tables t_i,t_j and t_k constitute a tree, where t_j is a supertable of t_i, t_k is a supertable of t_j and t_i is also an associated table. Hence, the initial $MCG(t_i)$ of t_i has been obtained in first step. Now we need to find the $MCG(t_j)$ and $MCG(t_k)$. In order to not violate the mapping consistency, i.e., every class of $MCG(t_i)$ must exist at least one class in $MCG(t_j)$ which is its superclass or the same class as it, $MCG(t_j)$ will be found by intersecting all superclasses of class of $MCG(t_i)$ and $MCG(t_i)$ itself. In the same way, $MCG(t_k)$ can be found since the initial matching result of $MCG(t_j)$ has been created. If the example becomes t_i and t_j are also associated tables, both of them have the initial $MCG(t_i)$ and $MCG(t_j)$ respectively. In this situation, we will revise $MCG(t_j)$ to a new $MCG'(t_j)$ by intersecting all superclasses of class of $MCG(t_i)$, $MCG(t_i)$ and $MCG(t_j)$. The following is the definition describing the processes of finding MCGs for two G/S tables t_i and t_j.

Definition 4.6.4. For a subtable t_i and its supertable t_j: If t_i has initial $MCG(t_i) = \{cls_1, cls_2, \ldots, cls_n\}$ and t_j doesn't.

- $MCG(t_j) = (SupC(cls_1) \sqcap SupC(cls_2) \sqcap \ldots \sqcap SupC(cls_n)) \sqcup MCG(t_i)$

If t_j has initial $MCG(t_j) = \{cls_1, cls_2, \ldots, cls_n\}$ and t_i doesn't.

- $MCG(t_i) = (SubC(cls_1) \sqcap SubC(cls_2) \sqcap \ldots \sqcap SubC(cls_n)) \sqcup MCG(t_j)$

If t_i and t_j both have the initial matching classes of group $MCG(t_i) = \{cls_1, cls_2, \ldots, cls_n\}$ and $MCG(t_j)$ respectively, a revised $MCG'(t_j)$ is

- $MCG'(t_j) = ((SupC(cls_1) \sqcap SupC(cls_2) \sqcap \ldots \sqcap SupC(cls_n)) \sqcup MCG(t_i)) \sqcap MCG(t_j)$

Note that $MCG(t_i)$ does not necessary to be revised in this case.

MCGs of Other Tables

Nevertheless, there are some G/S tables which can not satisfy the conditions described above, i.e., those G/S tables constitute a generalization/specialization tree but have no existing associated table from them. In these cases, their MCGs will be regarded as a universal classes in OWL ontologies and directly compute the similarities from all classes of OWL for these tables since these cases are rare and there is no other information can be further used. For those independent tables which are neither associated tables nor G/S tables, they will be also directly computed all possible similarities between them and all classes of OWL ontologies.

4.6.3 Mapping Tables and Non-Foreign Keys

After finishing second step, we will compute the similarities between table and its MCG. In phase one, the similarities between foreign keys and object properties comprises the local similarity and the external similarity. Different from the phase one, similarities between tables and classes in this phase are only composed of the local similarity because on the one hand, the external similarity for one table and one class is unnecessary since the external structure of every class of one MCG is no difference, and on the other, the internal similarity does not suit to be included into the similarity between one table and one class since our experiment had shown that the internal similarity does not really help the accuracy of the final mapping result.

4.6.4 Mapping Columns

In phase one, we map various kinds of foreign keys to object properties to get the features of MCGs for associated tables and in phase two, MCG for every table is found by using the related features of classes expected to be mapped and then the mapping results of tables are done by mapping each table to its MCG.

After finishing these two phase, we need to determine every column of table requires to be mapped to which property of the mapping result of table or mapped to nothing since not all columns of table can be migrated into instances of class. Once completing the mapping columns, we can apply these results to various application such as transferring instances of columns to instances of properties. There are two kinds of columns, which are foreign keys and non-foreign keys. For Base FKs and Part-Of FKs, which properties should be mapped has been determined in phase one and the mapping columns rules for these two kinds of FKs are described as follows.

Definition 4.6.5. For tables t_i and t_j, if c_i is a Base or Part-Of FK of t_i referring to t_j, op_i is the mapping result of c_i in phase one and cls_i and cls_j are the mapping results of t_i and t_j respectively in phase two, the mapping columns rules for c_i will be :

- Case 1: If the mapping result op_i comes from the level one matching, c_i must be mapped to op_i of cls_i since cls_j has no property op_i.
- Case 2: If the mapping result op_i comes from the level two matching, c_i must be mapped to op_i of cls_j since cls_i has no property op_i.

Apparently, for Base FKs, whose mapping results only come from the level one matching and hence they only can apply Case 1 to map columns. Furthermore, if these foreign keys can map to object properties, they will become the range classes of object properties. For example, supposing that a table "railroad(trainNO, type, startStation, endStation)" in the relational database schema 5.3 is mapped to a class "RailwayLine", which has object properties "transEndTo" and "transStartFrom" and both of them describe some transportation line will has its start station and end station. Since columns of railroad "startStation" and "endStation" are foreign keys and have the same meaning as "transEndTo" and "transStartFrom", they will be mapped to these two object properties respectively in phase one. In this example, also meaning that "startStation" and "endStation" will also become the range classes of "transEndTo" and "transStartFrom".

For other columns including IS-A FKs and non-foreign keys, which properties they should be mapped depend on the similarities between them. Besides, these other columns can also be possible to be mapped to object properties if they can independently represent the range classes of object properties. If these other columns need to be matched to some datatype properties, in this time, their matching result will depend on the datatypes and similarities they have. The datatypes of columns in the relational database are build-in SQL datatypes. Similar to columns, datatype properties in OWL ontologies, which are not like object properties that have range classes, make use of the RDF datatyping scheme, which provides a mechanism for referring to XML Schema datatypes [29]. Besides, two entities will have almost no chance to match each other if their datatypes of range are different. For example, a datatype property named "Year" whose datatype is "time" and a none foreign key column called "Creator" whose datatype is "CHARACTER". Apparently Year and Creator represent different thing in the real world and hence they should not be the final mapping result even though both of them may appear a high similarity to some string-based matchers. Therefore, in order not to spend time on computing unnecessary similarities, we divide datatypes into three main groups based on the correspondences between SQL datatypes and XML Schema datatypes. These groups are in Table 4.2. For a column c_i of table t_i, its possible matching datatype properties of group will be only those which have the same datatypes as c_i.

Moreover, as opposed to properties in OWL ontologies, columns in the relational database can not be independent existing, they depend on their own table, i.e., when a table is deleted its columns is also disappeared. Hence, by taking advantage of three views as mentioned above:

- A column can be possible to be mapped to object properties.
- A column can only be mapped to the datatype properties which have the same dataytype as it.
- A column can only be mapped to properties of the mapping result of table which this column stores in.

Table 4.2: Groups of SQL and XML Schema Datatypes

Groups of Datatypes		
Group	SQL Datatypes	XML Schema Datatypes
Numeric Group	SMALLINT	short
	INTEGER	integer
	INTEGER	positiveInteger
	INTEGER	negativeInteger
	INTEGER	nonPositiveInteger
	INTEGER	nonNegativeInteger
	INTEGER	int
	INTEGER	long
	NUMERIC	decimal
	DECIMAL	decimal
	FLOAT	float
	REAL	float
	DOUBLE PRECISION	double
String Group	CHARACTER	string
	CHARACTER VARYING	string
Time Group	TIME	time
	TIME WITH TIME ZONE	time
	DATE	date
	TIMESTAMP	datetime
	TIMESTAMP WITH TIME	datetime

The similarity of the best matching candidate between a column c_i and a properties group of cls_i can be defined through as follows.

Definition 4.6.6. Let:

- c_i be a column excluding Base FK and Part-Of FK in table t_i and the datatype of c_i be *ctype*
- $GP_{dtype}(cls_i)$ =$\{dp_1, dp_2, \ldots, dp_n, op_1, op_2, \ldots, op_m\}$ be a set of properties including every datatype property with the same datatype *dtype* and every object property of cls_i.

If *ctype* is the same as *dtype* according to Table 4.2 and cls_i is the mapping result of

t_i, p_i is the best matching candidate property for c_i if :

$$Sim(c_i, p_i) = Sim(c_i, GP_{dtype}(cls_i)) = \max((\max_{1 \leq x \leq n} Sim_{local}(c_i, dp_x)) \sqcup (\max_{1 \leq y \leq m} Sim_{local}(c_i, op_y)))$$

Chapter 5

Prototype System: Annotator

We develop a prototype system called “Annotator” (see Figure 4.1) implementing the approach in Chapter 4. This prototype system is created as a plugin of Protégé[38] and integrated in “the Traveller”[39] as its component. The Traveller is a Semantic Web-based trip planning system that aims to help a customer or a travel agent to semantically discover, combine, and invoke the desired Web services for a trip by the help of related matching service. Web services are based on widely accepted standards such as WSDL (Web Service Definition Language) and UDDI (Universal Description, Discovery and Integration). However, just like the deep Web, the data used by Web Services usually comes from relational databases and hence Annotator will play a role to support the Traveller by mapping all related relational databases to the domain ontologies used by the Traveller. The functions of the Traveller can see Figure 5.1. In the following sections, we introduce how Anotator works.

5.1 Annotator

Annotator is a subsystem of the Traveller whose object is to assist system administrator to engage in mapping the relational databases of travel industry to travel-related OWL ontologies used by the Traveller. Figure 5.2 is the GUI of Annotator and this figure also show the mapping results between our relational database example 5.3 and the OWL ontologies used by the Traveller. This mapping process for domain experts can be described as follows.

- Experts loads the OWL ontologies of the Traveller into Protégé to use the functions of Protégé provideds to maintain ontologies.

- Now if expert want to map a relational database to the current OWL ontologies, expert only need to open the tab of Annotator and the OWL ontologies used by Protégé currently will automatically be loaded into the process of Anotator.

- System will begin to map the relational database to OWL ontologies once expert executes the related command. After finishing mapping and expert saves the mapping results, these mapping results will automatically reflect in the GUI of OWL ontologies.

Note that the information of mapping results of foreign keys and tables includes the similarity of these mapping results and when experts clicks these mapping results, system will show options for experts to revise mapping results and these options are sorted based on their ranks of similarities. But as for mapping results of columns, usually only small group of columns of tables require to be mapped to properties and besides, in contrast to names of properties, the ones of columns are often incomplete. Therefore, a low performance of mapping results for columns can be expected. For this reason, system will only show the best matching property in option one and other options will leave to experts to choose if they need to revise mapping results

5.1.1 Matcher

Matcher used in this system is a hybrid matcher combining Lin and I-Sub measures. Lin measure computes the similarity of two words in WordNet. How similar of these two words should be depends on their positions and information contents in the WordNet tree. For these reasons, we use a famous Java API called "Java WordNet Library (JWNL)" [1]. JWNL is a Java API for accessing the WordNet dictionary. With JWNL we can easily manipulate the related information of a word in WordNet such as deciding the relationship between two words in the WordNet tree. Once we can know the related position of two words in the WordNet tree using JWNL, LCS of these two words and their information contents could be found and computed.

[1]`http://jwordnet.sourceforge.net/`

Calculating the Lin similarity between two words needs extra time computing their information contents and this extra time will cause an inconvenience for uses since there require quite a lot information contents of words for our mapping system. Therefore, we use Java WordNet Similarity Library [2] to speed up the computing similarities. Java WordNet Similarity is a pure Java API implementation of a number of standard WordNet similarity measures include Lin and this API had computed in advance information content of every word in WordNet 2.0 stored in a file. Furthermore, there are repeated pair of strings which require to be computed the similarities especially for the ones between foreign keys and object properties when computing the similarities between entities and hence we create a repository storing the similarity of every pair of strings. Whenever a pair of words want to be computed system first looks up the repository to decide if this pair of words need to be computed the similarity or not, thereby, saving more time for users.

5.1.2 Relational Database

The related information of a relational database schema used as inputs in this system can be easily retrieved through Java Database Connectivity (JDBC) API. JDBC [3] is a database-independent connectivity between the Java programming language and a wide range of databases and hence we use JDBC as a mediator between our system and our goal relational database MySQL. We use JDBC to connect to MySQL database and make full use of "DatabaseMetaData" object in JDBC. With DatabaseMetaData we can get all metadatas of schema from relational databases.

5.1.3 OWL Ontology

Our system is developed as a plugin of Protégé[38]. Protégé is used to construct, edit, update, maintain and retrieve OWL ontology and created by Stanford University. Stanford University released not only the Protégé system but the Protégé-OWL API used to build Protégé. Protégé-OWL API is so powerful that we can easily manipulate OWL ontologies. For example, given one class cls_i, the set of superclasses or subclasses of cls_i

[2]`http://nlp.shef.ac.uk/result/software.html`

[3]`http://java.sun.com/javase/technologies/database/`

can be immediately obtained throug Protégé-OWL API and given one object property op_i the set of domain classes or range classes of op_i can be also immediately obtained.

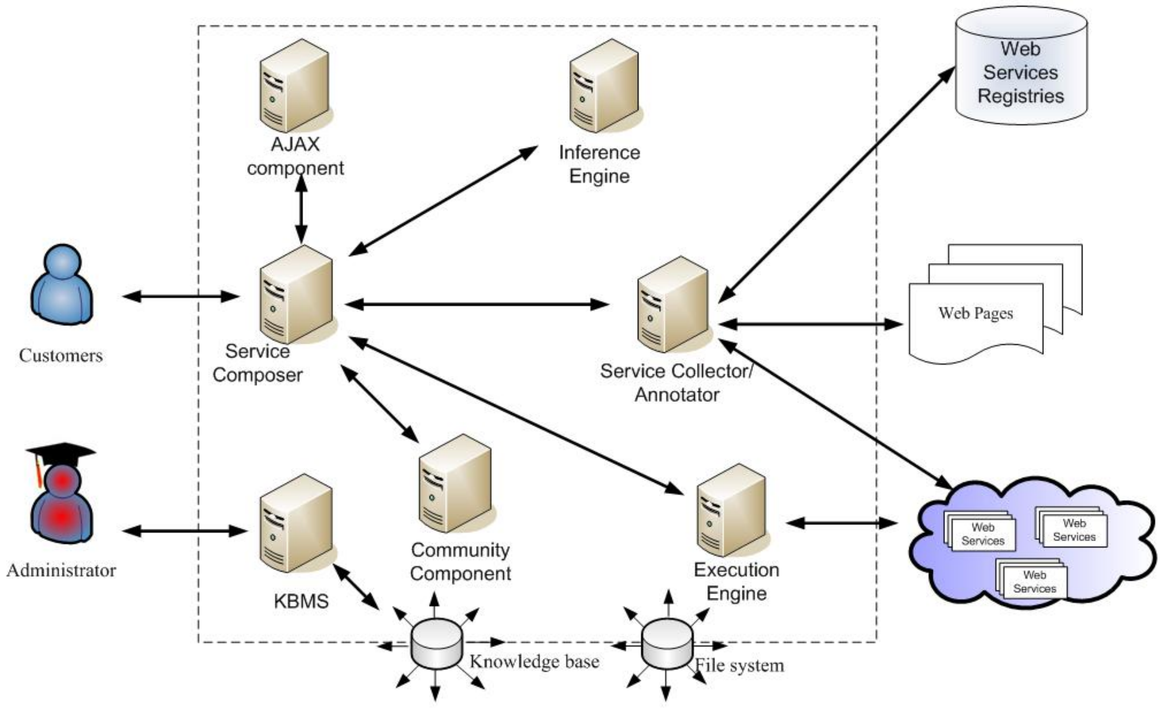
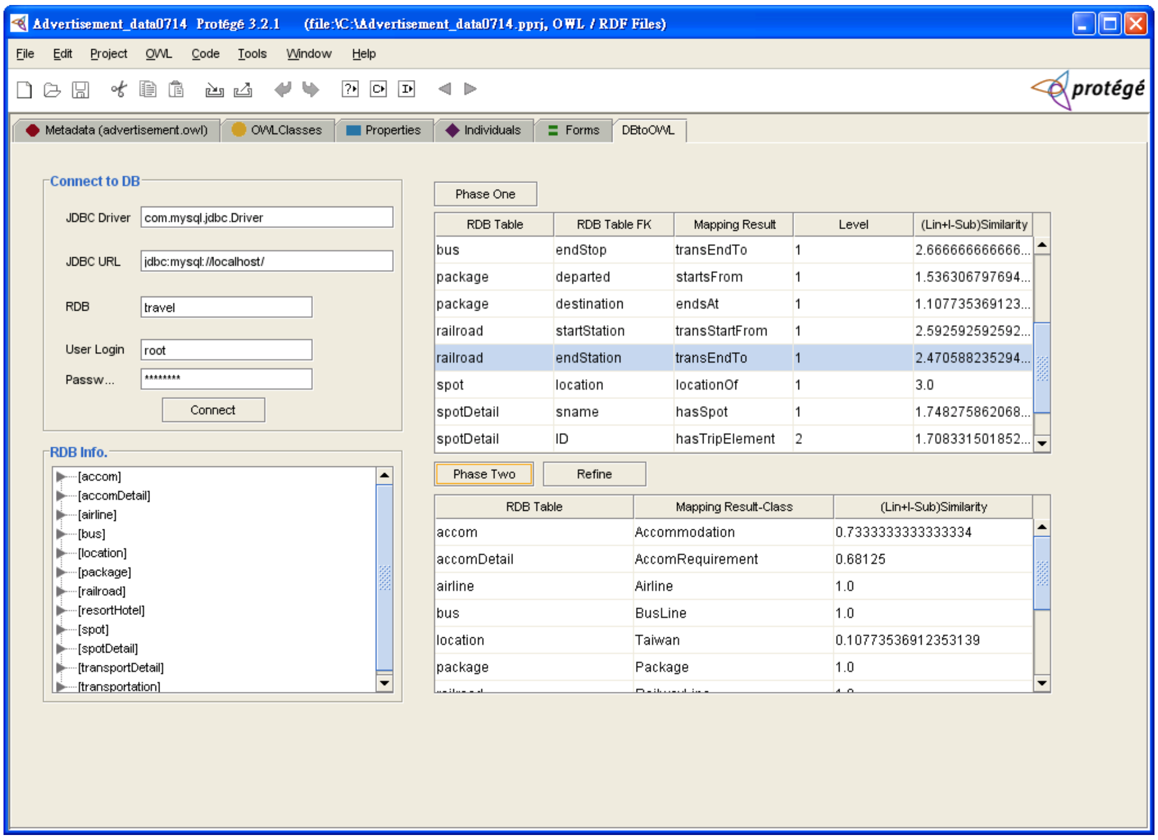


Figure 5.1: System Architecture of The Traveller



Advertisement_data0714 Protégé 3.2.1 (file:\C:\Advertisement_data0714.pprj, OWL / RDF Files)

Connect to DB

JDBC Driver	com.mysql.jdbc.Driver
JDBC URL	jdbc:mysql://localhost/
RDB	travel
User Login	root
Passw...	********

RDB Info.

- [accom]
- [accomDetail]
- [airline]
- [bus]
- [location]
- [package]
- [railroad]
- [resortHotel]
- [spot]
- [spotDetail]
- [transportDetail]
- [transportation]

RDB Table	RDB Table FK	Mapping Result	Level	(Lin+I-Sub)Similarity
bus	endStop	transEndTo	1	2.666666666666...
package	departed	startsFrom	1	1.536306797694...
package	destination	endsAt	1	1.107735369123...
railroad	startStation	transStartFrom	1	2.592592592592...
railroad	endStation	transEndTo	1	2.470588235294...
spot	location	locationOf	1	3.0
spotDetail	sname	hasSpot	1	1.748275862068...
spotDetail	ID	hasTripElement	2	1.708331501852...

RDB Table	Mapping Result-Class	(Lin+I-Sub)Similarity
accom	Accommodation	0.7333333333333334
accomDetail	AccomRequirement	0.68125
airline	Airline	1.0
bus	BusLine	1.0
location	Taiwan	0.10773536912353139
package	Package	1.0

Figure 5.2: Annotator

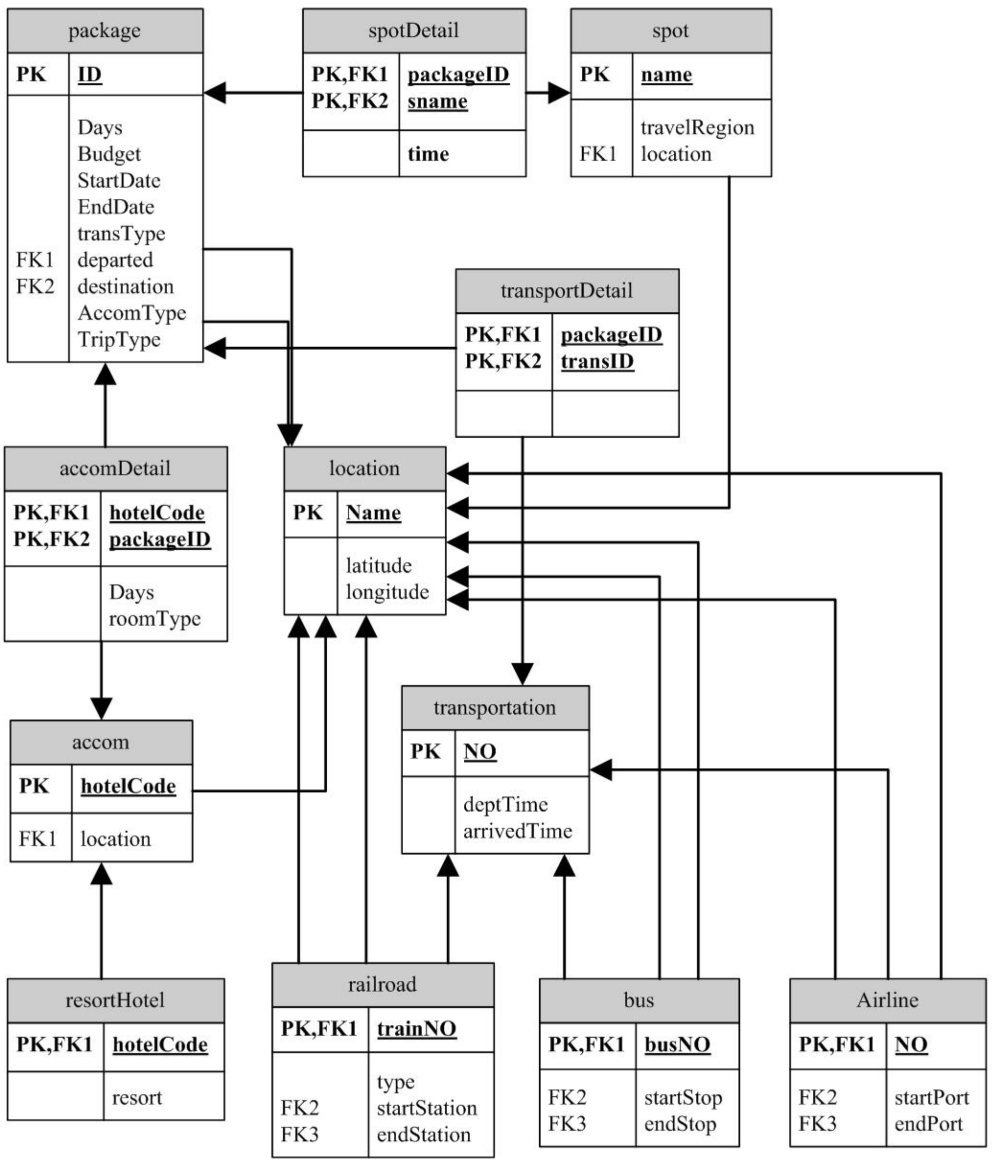


Figure 5.3: Example: Relational Database Schema

Chapter 6

Conclusion

We implemented a system of mapping relational databases to OWL ontologies using the approach proposed in this thesis. With this system, domain-related experts can semiautomatically engage in mapping various sources of relational databases to OWL ontologies such as the underlying relational databases of dynamic Web pages or the ones used by Web services. Mapping processes of this system are composed of two phases and we detail our contributions and future works as follow.

6.1 Contributions

- **Construct an implicit matching level between foreigns keys and object properties**: We reach the object of semantic enrichment by classifying tables and columns of the relational database. As far as we know, many approaches to directly mapping a relational database to OWL ontologies do not use the concept of semantic enrichment, i.e., extracting the implicit semantic metadata expressed in the relational databases. Hence, the mapping results of these approaches are often the relational-like ones. Besides, most of the approaches only adapt the rule of inheritance relationship between tables to map G/S tables to classes. Apparently this is insufficient since the inheritance relationship in the relational database is rare case and there still exist other implicit semantic which is not extracted. However in our mapping system, we not only take the Generalization/Specialization relationship between tables into consideration but also take advantage of the inverse relationship between tables which are implicitly expressed in a relational database to construct an implicit matching level between foreigns keys and object properties.

- **Devise a new method using the concept of cluster analysis**: We devised a new method to map tables of a relational database to classes of OWL ontologies based on the concept of cluster analysis. The object of using this concept is to find the MCG for every table. Finding the MCG for every table has at least four advantages. First, most of the time in mapping problem is spent on computing the similarities between entities and hence if we do not find the MCG for tables, one table will usually match to every class of OWL ontologies that is equal to compute every similarity between one table and every class of OWL ontologies. However, we can reduce this problem to only compute the necessary similarities after finding the MCGs for associated and G/S tables. Second, for one table, its MCG means we collect all classes with the same features as the one that should be the final mapping result in the real world, in other words, every class of MCG has share some same properties with the real final mapping result. Therefore, sometime we can avoid resulting a great error of mismatching even though the users choose or system computes the wrong mapping results. Third, since the final mapping result must be in the MCG of the table, the exact matching class which should be the final mapping result in the real world must be included into MCG even though the names of tables and classes are greatly divergent. Finally, in phase one, the total similarities between foreign keys and object properties include the external similarities and these ones in fact are right the local similarities between tables and classes in phase two. Therefore, users can find that there is almost no waiting time to see the matching results in phase two because most of the local similarity between tables and classes has been computed in phase one.

- **Create a flexible approach**: The strategy what the similarity should be composed of can depend on the situation when computing the similarity between one foreign key and one object property. For example, in worse case, the composition of similarity can combine the local similarity, the internal similarity and the external similarity if the names of terms in relational databases is disorder a lot. On the contrary, in best case, the composition of similarity can be only the local similarity.

6.2 Future Work

- **Need a more powerful matcher**: Many matchers have been proposed so far. However, there is still lacking of matchers specifying for mapping between relational databases and OWL ontologies, i.e., most of them only deal with mapping between databases or ontologies. A matcher specifying for mapping between databases is not fit to our approach because computing similarities between relational databases and OWL ontologies is usually from a less ordered names of terms to a more ordered ones. Therefore, our approach using a hybrid matcher combining Lin and I-Sub measures. However, these two measures are sensitive to the appearance of string of database since Lin and I-Sub specify for mapping between ontologies, i.e., if two strings have some degree of similarity but due to the reason one of them cannot be looked up in WordNet and both of them have no substring relation, the similarity of them will be zero and this will indirectly require more feedback from users. For this purpose, we need to devise or find a more powerful matcher that devotes to database-to-ontology mapping.

- **Find more implicit information**: We take advantage of the semantic enrichment in phase one and its object is reached by simply using the classifying tables and columns. Nevertheless, if only simply considering the classification, there still exist some implicit semantic metadatas cannot be captured for some tables especially for those may express more than one entity in the real word. For instance, Johannesson [21] proposed that if one table has more than one candidate key, this table may be possible to represent two different entities in the real word. Therefore, in order to deal with such this case, more complex sematic enrichment may be necessary.

- **A more sophisticated way to find MCG for every table is required**: We find MCG for every table and every member of MCG satisfies the mapping consistency. For our example 5.3, tables “package”, “spotDetail” and “spot”, all of them have non-null set of MCG. The fact of relational database shows “package” and “spotDetail” have binary relationship and “spotDetail” and “spot” also have binary relationship. Their non-null set of MCG must be satisfy mapping consistency because no matter which class of MCG is chosen as a final mapping result, these results can always relate each other through the mapping result in phase one. In this example, if classes “Package”, “SpotAdvertisement” and “Spot” are the final mapping results of these tables and object properties “hasTripElement” and “hasSpot” are the mapping result of foreign keys “packageID” and “sname” in “spotDetail”. Therefore, the mapping class “Package” can relate to “SpotAdvertisement” through “hasTripElement” and the mapping class “SpotAdvertisement” can relate to “Spot” through “hasSpot”. However, if one of these tables whose MCG is a null set, mapping consistency will be violated. For example, if the MCG of “spotDetail” is a null set, i.e., there is no mapping result for “spotDetail” and it will cause that the mapping result of “package” cannot be related to mapping result of “spotDetail” through “hasTripElement” , indirectly, violating the mapping consistency. Therefore, if we want to also analyse this null case of violating the mapping consistency, a more complex way to find MCG for every table is required.

Bibliography

[1] Y. An, A. Borgida, and J. Mylopoulos. Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. *International Semantic Web Conference.*, pages 6–20, 2005.

[2] I. Astrova. Reverse engineering of relational databases to ontologies. *Proceedings of 1st European Semantic Web Symposium, Heraklion, Crete, Greece, LNCS*, 3053:327–341, 2004.

[3] I. Astrova. Reverse engineering of relational databases to ontologies: An approach based on an analysis of html forms. *Proceedings of the 1 stEuropean Semantic Web Symposium*, pages 327–341, 2004.

[4] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. *Proceedings of the International Conference on Management of Data*, 2005.

[5] J. Barrasa, O. Corcho, and Gomez-Perez. R2O, an extensible and semantically based database-to-ontology mapping language. *Second Workshop on Semantic Web and Databases*, 2004.

[6] A. Behm. Migrating relational databases to object technology. 2001.

[7] T. Bemers-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 279(5):34–43, 2001.

[8] S. Benkley, J. Fandozzi, E. Housman, and G. Woodhouse. Data element tool-based analysis (delta). *MITRE Technical Report MTR*, 1995.

[9] S.M. Benslimane, D. Benslimane, M. Malki, Y. Amghar, and H. Saliah-Hassane. Acquiring OWL ontologies from data-intensive web sites. *Proceedings of the 6th international conference on Web engineering*, pages 361–368, 2006.

[10] M. K. Bergman. The Deep Web: Surfacing Hidden Value, 2000.

[11] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting context into schema matching. *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 307–318, 2006.

[12] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[13] H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. *Proceedings of the Very Large Data Bases Conference*, pages 610–621, 2001.

[14] A.H. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. *Proceedings of the eleventh international conference on World Wide Web*, pages 662–673, 2002.

[15] D. Dou, P. LePendu, S. Kim, and P. Qi. Integrating databases into the semantic web through an ontology-based framework. *International Workshop on Semantic Web and Database*, pages 33–50, 2006.

[16] F. Giunchiglia, P. Shvaiko, and Yatskevich M. S-Match: an algorithm and an implementation of semantic matching. *In Proceedings of ESWS'04.*, 2004.

[17] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[18] S. Handschuh, S. Staab, and R. Volz. On deep annotation. *Proceedings of the twelfth international conference on World Wide Web*, pages 431–438, 2003.

[19] W. Hu and Y. Qu. Discovering simple mappings between relational database schemas and ontologies. *International Semantic Web Conference.*, pages 225–238, 2007.

[20] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 1989.

[21] P. Johannesson. A method for transforming relational schemas into conceptual schemas. *Proceedings.10th International Conference*, pages 190–201, 1994.

[22] M.R. Koivunen and E. Miller. W3C Semantic Web activity. *Semantic Web Kick-Off in Finland*, pages 27–44, 2001.

[23] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *English translation: Sov. Phys.—Dokl., vol. 10, no.8*, pages 707–710, 1966.

[24] M. Li, X.Y. Du, and S. Wang. Learning ontology from relational database. *Proceedings of the fourth international conference on machine learning and cybernetics*, 6:3410–3415, 2005.

[25] D. Lin. An information-theoretic definition of similarity. *In Proceedings of the 15th International Conference on Machine Learning*, 1998.

[26] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. *Proceedings of the Very Large Data Bases Conference*, pages 49–58, 2001.

[27] A. Maedche and S. Staab. Ontology learning for the Semantic Web. *IEEE Intelligent Systems and Their Applications*, pages 72–79, 2001.

[28] A. Mark. Pure Java WordNet similarity library. `http://nlp.shef.ac.uk/result/software.html`.

[29] D.L. McGuinness, F. van Harmelen, et al. OWL web ontology language overview. *W3C Recommendation*, 2004.

[30] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and itsapplication to schema matching. *In Proceedings of ICDE*, pages 117–128, 2002.

[31] G.A. Miller et al. WordNet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[32] W. Nathan. Google at how many billions. `http://google.blognewschannel.com/archives/2005/01/23/google-at-how-many-billion-9-11/`.

[33] N.F. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 2004.

[34] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. pages 448–453, 1995.

[35] M. Steinbach, P. N. Tan, and V. Kumar. *Introduction to Data Mining.* Pearson Addison-Wesley, 2005.

[36] G. Stoilos, G. Stamou, and S. Kollias. A string metric for ontology alignment. *International Semantic Web Conference*, pages 624–637, 2005.

[37] Q. Trinh, K. Barker, and R. Alhajj. RDB2ONT: A tool for generating OWL ontologies from relational database systems. *Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, pages 170–170, 2006.

[38] Standford University. Protégé. `http://protege.stanford.edu`.

[39] Te-Wei Yang. Web services search and composition by combining Web 2.0 and Semantic Web technology. Master Thesis, National Taiwan University, 2008.