CHAPTER 1:
INTRODUCTION
Chapter 1: Introduction

1.1. Background

The web is the platform for information publishing; it is the biggest source of information of any type. There are a lot of valuable data and business data on the web that organizations can use to improve their decision making process. It is therefore, very important and critical that these information are complete, precise and timely (Fensel et al., 1999; Wang and Kim, 2002). It is also vital that such external information be systematically managed and utilized for users. Each information system on the web is modeled and implemented differently according to the requirements of the application domain. The access, retrieval and utilization of information from the different data sources imposes a need for the data to be integrated because there are many types of heterogeneity and differences among web sources that makes a combined effort to access data from different sources on the internet difficult and error-prone (Kashyap and Sheth, 1998; Fensel, 2001a, Fensel et al, 2001). The following HTML pages from different room reservation systems, as shown in Figures 1.1 & 1.2, are a simple illustration of this. For example, if a user queries rates of hotel rooms, the retrieval and combination of data related to rates of rooms in the following HTML pages fail because they have heterogeneity conflicts such as: using different names (“price” and “Rate”) and different units (“EUR” and “USD”) to represent cost of rooms.

<table>
<thead>
<tr>
<th>Select</th>
<th>Hotel</th>
<th>type</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gurney</td>
<td>single</td>
<td>EUR 67.99/day</td>
</tr>
<tr>
<td></td>
<td>Rose</td>
<td>double</td>
<td>EUR 75.99/day</td>
</tr>
</tbody>
</table>

Fig. 1.1 Reservation system A
The solution to the problem mentioned above is a web data integration system. External information can be extracted from web sources and utilized for users through a web data integration system (Chen et al., 2000; McCann et al., 2003). The design of such a system is not easy because the differences in web data make data integration a difficult process. Integration of heterogeneous data sources from the internet is a complex activity that involves reconciliation of various levels of conflicts (e.g., different data model, different syntax and semantic among web sources).

Two common approaches have been proposed to building data integration systems (Widom, 1995). The first approach is called the virtual approach (shown in Figure 1.3). A user or an application system poses a query to the system. The data integration system accepts the query, determines which set of information sources are able to answer the query and generates the appropriate query plans for each information source. The data integration system performs the appropriate translation, filtering and merging of the information and returns the final answer to the user or application (Ibrahim and Schwinger, 2001).

The second approach is called the materialization approach to data integration (shown in Figure 1.4). Information from each particular source that is of interest to specific users or applications is extracted in advance, translated, filtered and merged with relevant information from other sources and stored in a centralized repository. When a user or an

---

Fig. 1.2 Reservation system B

The solution to the problem mentioned above is a web data integration system. External information can be extracted from web sources and utilized for users through a web data integration system (Chen et al., 2000; McCann et al., 2003). The design of such a system is not easy because the differences in web data make data integration a difficult process. Integration of heterogeneous data sources from the internet is a complex activity that involves reconciliation of various levels of conflicts (e.g., different data model, different syntax and semantic among web sources).

Two common approaches have been proposed to building data integration systems (Widom, 1995). The first approach is called the virtual approach (shown in Figure 1.3). A user or an application system poses a query to the system. The data integration system accepts the query, determines which set of information sources are able to answer the query and generates the appropriate query plans for each information source. The data integration system performs the appropriate translation, filtering and merging of the information and returns the final answer to the user or application (Ibrahim and Schwinger, 2001).

The second approach is called the materialization approach to data integration (shown in Figure 1.4). Information from each particular source that is of interest to specific users or applications is extracted in advance, translated, filtered and merged with relevant information from other sources and stored in a centralized repository. When a user or an
application system poses a query, the query is evaluated directly at the repository without accessing the original information sources (Ibrahim and Schwinger, 2001).

In these two approaches, the system needs to integrate data from different sources. The integration of semi-structured data from different sources on the internet is a complex process because of existing data heterogeneity among web sources. Resolving this heterogeneity is a major obstacle in web data integration systems (Bornhovd and Buchmann, 2001).

In this chapter, Section 1.2 clarifies the importance of the semantic heterogeneity problem in data integration systems. We focus on semantic conflicts in schema level because the aim of the thesis is in resolving this type of conflicts. Section 1.3 explains why ontologies are suitable as a solution to semantic heterogeneity problems (ontologies are discussed in full detail in Chapter 3). Sections 1.4 and 1.5 present the scope and objective of the thesis. Section 1.6 describes the research methodology of the thesis and finally Section 1.7 briefly introduces the remaining chapters of the thesis.
1.2. Description of Problem

Organizations need data from many external sources in order to improve their decision making process. One of these external sources is the web. Data on the web are heterogeneous. In order to use these heterogeneous data from the internet we need to integrate them. The integration of heterogeneous data sources from the internet is a complex activity that involves reconciliation of various levels of conflicts. Before we can integrate the heterogeneous data we need to resolve these heterogeneity conflicts. There are different views about classification of Heterogeneity conflicts. Heterogeneity conflicts can be classified according to the following abstraction levels (shown in figure 1.5) (Kim and Seo, 1991 and Fileto, 2001):

- **Data Value Conflicts**: Data value conflicts are those conflicts that arise at the instance level. They are related to the representation or the interpretation of the data values. Examples of these conflicts are discrepancies of type, unit, precision and allowed values (“kg” and “gram” or “$” and “dollar”).

- **Schema Conflicts**: Schema conflicts are due to different alternatives provided by one data model to develop schemas for the same reality. For example, what is modeled as an attribute in one relational schema may be modeled as an entity in another relational schema for the same application domain (Author as attribute for the entity of book and author as an entity that has a relationship with book). Another example two sources may use different names to represent the same concept (“price” and “cost”) , or the same name to represent different concepts , or two different ways, for conveying the same information (“date of birth” and “age”).
• **Data Model Conflicts**: Data model conflicts occur when databases use different data models, e.g., one database designed according to the relational model, and another one object-oriented.

Conflicts in each level can be categorized into two categories:

• **Syntactic Conflicts**: Syntactic conflicts refer to discrepancies in the representation of data (“1/23” and “1.23” or “price=23$” and “price: 23$”).

• **Semantic Conflicts**: Semantic conflicts refer to disagreement about the meaning, interpretation use of the same or related data (“staff” and “employee”).

![Fig. 1.5 Data heterogeneity levels](image)

In this research, our focus is over semantic heterogeneity in data schema level. We call semantic schema heterogeneity or semantic schema conflict (shown in dashed circle of Figure 1.5). We briefly explain semantic schema heterogeneity in the next sub-section and in detail in Chapter 3.

**1.2.1 Semantic schema heterogeneity**

Semantics is people’s interpretation of data according to their understanding of the world (Sowa, 2002). Semantic at the schema level is defined as what determines how the terms and concepts are associated with things in the application domain (Sheth, 1998). It is related to the meaning of schema elements (e.g., classes, attributes and relationships). Any user or application has its own semantics based on requirements and common sense from
the application domain. We refer to different interpretations of data as semantic heterogeneity and if such differences are related to schema elements we refer to those as semantic schema heterogeneity (Geppert and Hakimpour, 2001).

In this research, semantic conflicts at the schema level, will be thoroughly investigated and studied (please refer to Chapter 3). The aim of our work is to give a solution for resolving semantic schema heterogeneity in a web data integration system.

1.3. **Solution for Problem**

The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, is a catalog of the types of things that are assumed to exist in a domain (Sowa, 2002). The word ontology comes from the Greek *ontos* (being) and *logos* (word). It has been introduced in philosophy in the 19th century, by German philosophers, to distinguish the study of being as such from the study of various kinds of beings in the natural sciences (Guarino, 1998). As stated in (Gruber, 1993): “An ontology is a formal, explicit specification of a shared conceptualization.”

Ontologies help to figure out what a specific term means. The ontology provides a way to describe the meaning and the relationships of terms so that a shared understanding or a consensus is acquired among people and machines (Guarino, 1998 and Gruber, 1993).

The ontology is an effective and notable solution to semantic heterogeneity problem in web data integration. Explicit semantic information of terms in ontology can help in resolving semantic heterogeneities among web data. To resolve semantic heterogeneity we need the consensus of the community over the semantic of terms. This consensus is realized through the ontology (Hakimpour, 2003). Ontologies have attracted the attention
of many researchers as a means of resolving data heterogeneity problems (Hakimpour, 2003). The ontology is also becoming an effective means of intelligent information integration, information interoperability, information retrieval, electronic commerce, and knowledge management (Fensel, 2001a). In computer science, ontologies were developed in Artificial Intelligence to facilitate knowledge sharing and reuse. The reason ontologies are becoming so popular is in large part due to what they promise: a shared and common understanding of some domain that can be communicated between people and application systems (Fensel, 2001a; Fensel, 2001b).

In our work, we use ontologies for resolving semantic conflicts among web sources and propose a system architecture based on the ontology. In our approach, web sources are assumed to have their own ontology. That means each term in one web source has a semantically corresponding term in its ontology and the meaning of the term is the same with the relevant term’s definition in the ontology. Therefore any community is free to choose its own vocabulary and semantic, independent from other communities’ based on their requirements and common sense from the application domain. By using ontologies, communities are free to communicate based on their own defined ontologies (Hakimpour, 2003).

Independency of communities in defining their ontologies may cause a major problem. We need the consensus of the community over the semantic of terms for resolving semantic heterogeneity problem. This consensus in web data context is not feasible because communities are free to use their own vocabularies and semantics according to their requirements and common sense. Therefore any web source has its own ontology that its community of users has agreed with all the defined terms in the ontology. This problem is solved by semantic mapping between ontologies (Hakimpour, 2003). This research proposes an algorithm to detect semantically corresponding terms between ontologies for
semantic mapping process. Our proposed approach uses domain specific ontologies for the creation of user queries. There is a domain specific ontology for each application area that covers the semantic definition of terms which are required for user queries in a particular application domain. The algorithm searches domain ontology in order to find user query concept and its query attributes through query path. The proposed mapping algorithm uses query path as a technique to enhance the quality of the mapping results and reduce the runtime of the algorithm. The query path gives two strength points to proposed mapping algorithm as follows:

1. *Reduce runtime of each achievement of mapping results:* It directs algorithm toward query concept and its attributes and causes to reduce the search domain of algorithm.

2. *Gain higher quality mapping results:* The query path possesses concepts which have some semantically relation with query concept. Therefore the algorithm has further information about meaning of query concept that helps to find corresponding term with query concept with higher quality mapping results.

---

1.4. **Outcome of Research**

The scope of our research is to give a solution for resolving semantic schema heterogeneity in a web data integration system. In order to achieve this goal, we focus on two major tasks as follows:

1. Propose an approach and system architecture for web data integration system: We propose an approach and system architecture for integration of heterogeneous web
data sources (please refer to Chapter 4). The proposed web data integration system relies on the ontology technology for resolving of semantic schema heterogeneity.

2. Propose an algorithm for semantic mapping between ontologies: In the second step, we focus over semantic mapping component of the proposed web data integration system and propose a semantic mapping algorithm for finding corresponding terms and reconciliation of semantic conflicts between web ontologies. We will show that, this algorithm resolves semantic schema heterogeneity between web data sources (please refer to Chapter 5).

1.5. Research Objectives

The main goal of this research is to present an approach for resolving semantic heterogeneities in schema level from web sources. In order to achieve this goal we must pursue the following important issues:

1. Clarify and define semantic schema conflicts (semantic heterogeneities at the schema level): Semantic schema conflicts must be defined and understood clearly. We must clarify semantic schema conflicts arising from the integration of heterogeneous data from the internet. We deal with this issue in Chapter 3.

2. Formalization of semantics through ontology: The formalization of web data semantics aim at structuring the meaning of data according to properties that can be automatically used by the web data integration system. We use ontology to add semantic to web data sources. The formalism of ontology provides the capability to express semantics. We deal with this issue in Chapter 3.
3. Propose a web data integration system: We propose an approach and system architecture for the integration of web data sources. We investigate and design the system architecture for resolving data heterogeneity levels between web data sources focusing on semantic schema heterogeneity.

4. Introduce an algorithm for semantic mapping between web ontologies: This algorithm finds semantic corresponding terms among the web ontologies and maps them to each other. We call this algorithm “semantic ontology mapping algorithm”. Semantic mapping between web ontologies is a major task in a web data integration process which helps to resolve semantic schema heterogeneity between web data sources.

1.6. Research Methodology

The methodology of the thesis is as following (shown in Figure 1.6):

First phase: Planning

A.1) Review relevant literature concerning research area

A.2) Define research problem, scope, outcome and objectives

First phase: Analysis

B.1) Review data heterogeneity focusing on semantic heterogeneity;

B.2) Review relevant literature concerning the background of categories and structure of data integration approaches focusing on those presented approaches for resolving semantic schema heterogeneity;

B.3) Review semantic web technologies (XML, RDF, RDF schema ...);
Second phase: Solution

C.1) Study Ontology and relevant issues of that as solution;

Third phase: Design

D.1) Propose and design an ontology-based web data integration system;

D.2) Propose and design an algorithm for finding semantic correspondence among the web ontologies for the semantic mapping process;

Fourth phase: Evaluation

E.1) Develop a prototype for evaluation of proposed semantic mapping algorithm.

Fig. 1.6 Research methodology

1.7. Thesis Structure

In this section, an introduction to the thesis is provided. The background of the work, the main problems, the overall objectives, the way of working are described. The structure of the rest of the thesis is as following:

- **CHAPTER 2: Data Integration System and Ontology.** This chapter explains how ontologies help to resolve semantic heterogeneity in data integration system. This chapter reviews projects and research works in the domain of semantic integration with a focus on works based on ontologies.
• **CHAPTER 3: Semantics and Ontology.** In this chapter, we intend to clarify the definition of existing concepts and terms in the scope of the thesis. This chapter investigates the types of data heterogeneities and the definition of semantic schema heterogeneities. Subsequently, this chapter discusses ontology and the related web based issues. Finally it explains OWL as the latest standard and language for description of ontologies on the web that has been developed by the World Wide Web consortium.

• **CHAPTER 4: Ontology-based Web Data Integration System.** This chapter presents an approach and system architecture for web data integration system in order to resolve data heterogeneity conflicts.

• **CHAPTER 5: Semantic Ontology Mapping.** This chapter focuses on semantic mapping module in the proposed web data integration system and explains the semantic ontology mapping process. Subsequently a mapping algorithm for finding semantically corresponding terms between web ontologies is presented in this chapter.

• **CHAPTER 6: Implementation and assessment of Ontology Mapping Approach.** The technical details of the ontology mapping algorithm are illustrated through a prototype.

• **CHAPTER 7: Discussion and Conclusion.** An evaluation of the achievements is presented here. This chapter summarizes the abilities of the proposed solution. It also presents the possible future improvements along with the conclusions made.
CHAPTER 2: DATA INTEGRATION SYSTEM and ONTOLOGY
2.1. Introduction

This chapter introduces approaches and research works related to semantic integration. We pay special attention to approaches and works using ontologies because our approach is based on ontologies. Major outlines of our work are:

- An approach and system architecture for web data integration;
- An algorithm for semantic mapping between web ontologies. This algorithm helps semantic schema integration between web sources in the web data integration system.

We introduce popular approaches for data integration and semantic schema integration in Section 2.2 and 2.3 and then conclude which approaches are suitable for our solution. We divide the research works into two categories according to our outlines. The first category (please refer to Section 2.4) introduces projects concerned with ontology-base data integration. These projects address semantic problems in data integration system by using ontologies. The second category (please refer to Section 2.6) briefly introduces projects related to semantic ontology mapping. These projects address semantic mismatches between ontologies. We survey existing shortcomings among reviewed web data integration systems in Section 2.5. In Section 2.7 and 2.8 we study the characteristics and shortcomings of the semantic ontology mapping algorithms. We discuss ontology architectures in data integration systems in Section 2.9 and choose appropriate ontology architecture for our web data integration system. In Section 2.10 we present a comparison framework for the reviewed ontology-based web data integration systems. We conclude this chapter in Section 2.11.
2.2. Overview of the Data Integration Approaches

As stated in Chapter 1, two common approaches have been advocated to building data integration systems (Widom, 1995). The first approach is referred to as the **virtual approach** to data integration (please refer to Figure 1.3). A user or an application system poses a query to the system. The data integration system accepts the query, determines which set of information sources is capable of answering the query and generates the appropriate query plans for each information source. On obtaining the results from the information sources, the data integration system performs the appropriate translation, filtering and merging of the information and returns the final answer to the user or application (Ibrahim and Schwinger, 2001).

The second approach is called the **materialization approach** to data integration (please refer to Figure 1.4). Information from each particular source that may be of interest to specific users or applications is extracted in advance, translated and filtered as appropriate, merged with relevant information from other sources and stored in a (logically) centralized repository. When a user or an application system poses a query, the query is evaluated directly at the repository without accessing the original information sources (Ibrahim and Schwinger, 2001).

The materialization approach is not suitable for highly dynamic, open and distributed environment (such as the internet) because this approach is suitable when the schemata are not subject to frequent changes. In a frequently changing and dynamic environment we are obliged to frequently modify the centralized repository and this is highly costly. Therefore, we follow the virtual approach for the design of our architecture. This approach is
beneficial and suitable for highly dynamic, open and distributed environment such as internet.

2.3. Overview of the Semantic Schema Integration Approaches

In the above two approaches, the system needs to integrate data from different data sources. Resolving semantic conflicts at the schema level is a major obstacle and a problem in the integration process. Two popular approaches have been proposed to resolve semantic schema level conflicts (Hakimpour, 2003). The first approach is *generation of an integrated global ontology or schema* (shown in Figure 2.1), which defines the structure of the data in a federated database system. The integration method must find out if the schema elements from different schemata refer to the same set of entities, or whether they are different and if they are different, to what degree they differ. Similarity relations are defined in order to find out whether and how elements from different schemata are semantically related. A reasoning system is used to merge two schemata based on the detected similarity relations. This approach is suitable when the schemata are not subject to frequent changes (Hakimpour, 2003).

In contrast to the approach suggested above, there is another popular approach. In this approach, a reasoning system finds the similarities between concepts in two schemata and the mediator maps the corresponding terms in two schemata (shown in Figure 2.2). The mediator uses a reasoning system to find the mapping between schemata.
This approach (shown in Figure 2.2) is suitable whenever the schemata are subject to frequent changes (such as DTDs in XML data), or when many data sources are involved, or the number of involved data sources change frequently (such as data sources on the Internet). In these examples, integration of schemata is not beneficial due to the frequent changes. Therefore, queries are translated for the component data sources and the response to the query, in turn, is translated on-the-fly. One drawback of this approach is the high processing cost, since for every query the schemata must be processed to derive the required mappings (Hakimpour, 2003). Therefore, we follow the second approach (shown in Figure 2.2) for our solution in resolving semantic conflicts that is suitable for highly dynamic, open and distributed environment such as internet.

### 2.4. Web Data Integration Projects

This section introduces some of the major projects that focus on web data integration. We categorize studied projects to ontology-based and non-ontological web data integration
system. Our focus is further on ontology-based projects for resolving semantic conflicts. Our proposed system architecture (please refer to Chapter 4) for web data integration is based on the results of these works.

### 2.4.1 Ontology-based web data integration projects

The following projects use ontologies for resolving of semantic heterogeneity conflicts between web data sources. We briefly explain them in the next subsections.

1. **COIN.** The COIN (Goh et al., 1999) project presents a suitable architecture for semantic interoperability between distributed information sources. A context interchange (COIN) mediator is an automated reasoning engine to assist an organization in resolving semantic conflicts between its own receiver’s context and the contexts of data sources. Context mediator resolve semantic conflicts among information sources through comparison of contexts associated with the information sources. The COIN framework uses a data model and logical language to define the domain model of the application and the contexts. The domain model has the role of the ontology in the COIN-framework.

   Context mediator in Coin-architecture performs the process of rewriting queries posed in the receiver’s context into a set of mediated queries where all potential conflicts are explicitly resolved. This process is according to the statements in the different contexts involved, what information is needed to answer the query and what and how conflicts may be resolved (Goh et al., 1999).

2. **MOMIS.** MOMIS (Mediator Environment for Multiple Information Sources) (Beneventano et al., 2004) is one approach to the integration and query of semi-structured and structured heterogeneous data sources. The goal of MOMIS is to define a global schema that allows uniform and transparent access to the data stored in a set of semantically heterogeneous sources. MOMIS creates a global virtual view (GVV) of
information sources, independent of their location or their data’s heterogeneity. MOMIS builds ontology through five phases as follows (Choi et al., 2005):

1) Local source schema extraction by wrappers;

2) Local source annotation with the WordNet;

3) Common thesaurus generation: relationships of inter-schema and intra-schema knowledge about classes and attributes of the source schemas.

4) GVV generation: A global schema and mappings between the global attributes of the global schema and source schema by using the common thesaurus and the local schemas are generated.

5) GVV annotation is generated by exploiting annotated local schemas and mappings between local schemas and a global schema.

MOMIS generates mappings between global attributes of the global schema and source schemas. For each global class in the global virtual view (GVV), a mapping table (MT) stores all generated mappings.

3. KRAFT. The KRAFT (Knowledge Reuse And Fusion/Transformation) system (Visser et al., 2000) is a research prototype which is being developed collaboratively with BT by three UK universities - Aberdeen, Cardiff and Liverpool - for combining and transforming constraint based knowledge.

The KRAFT architecture is designed to support knowledge fusion from distributed, heterogeneous databases and knowledge bases (Choi et al., 2005). Components of the KRAFT architecture help applications and users to:

- locate data and knowledge relevant to their current needs;
- combine and refine data and knowledge in order to generate the information they require;
• Identify and exploit the processing engines to solve their problems.

The KRAFT architecture will help information providers and system developers to:

• make their resources available to the widest population of users;
• cope with changes in the information or service they provide, while minimizing the effects on client applications;
• Keep account of users’ access to their resources (e.g. for billing).

KRAFT is a project for the integration of heterogeneous information, using ontologies to resolve semantics problems. They extract the vocabulary of the community and the definition of terms from documents existing in an application domain. KRAFT detects a set of ontology mismatches and establishes mappings between the shared ontology and local ontologies.

4. OBSERVER. OBSERVER (Mena et al., 2000) is an approach for query processing in global information systems based on interoperation across pre-existing ontologies. OBSERVER allows users to pose their queries by using ontologies against heterogeneous data sources. It replaces terms in user queries with corresponding terms in the data source ontologies by detecting any similarities between the user ontology and the local ontology. OBSERVER uses Description Logic as an ontology definition language and a query language.

An interesting module in OBSERVER architecture is the Inter-Ontology Relationships Manager. It keeps the similarities between ontological definitions of terms in different ontologies. By means of such inter-ontology relations, OBSERVER replaces terms in user queries with corresponding terms in the local ontologies. OBSERVER merges ontologies based on existing similarity relations kept in the Inter-ontology Relationships Manager. Subsequently, it performs semantic mapping between two ontologies. The major problem
in OBSERVER is detecting the similarities among ontologies and performing semantic mapping. OBSERVER tries to resolve this problem by Inter-Ontology Relationships Manager module.

5. SIMS. SIMS (Arens et al., 1997) is a system that exploits a semantic model of a domain to integrate the information from various information sources. SIMS uses a model from an application domain that includes a hierarchical terminological knowledge. The application model plays the role of a global ontology in SIMS. SIMS uses a model from each information source that must be described for this system by relating the objects of each source to the global domain model. The relationships clarify the semantics of the source objects and help to find semantically corresponding (similar in meaning) objects. SIMS uses Loom as a knowledge representation language to describe the domain model and contents of the information sources. Loom is a knowledge representation languages developed in the Artificial Intelligence research group at the University of Southern California's Information Sciences Institute. In SIMS the user formulates a query in terms of the ontology. Then SIMS reformulates the global query into sub-queries for each appropriate source, collects and combines the query results, and returns the results (Arens et al., 1997).

2.4.2 Non ontological web data integration projects

The following researches do not offer any ontological layers for the integration of heterogeneous web data. We briefly explain them in the next sub-sections.

1. TSIMMIS. TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources (Chawathe et al., 1994) is a system that facilitates the rapid integration of heterogeneous data sources. TSIMMIS is a federated system based on mediated-wrapper architecture that does not offer an ontological layer (Chawathe et al., 1994). The data integration TSIMMIS architecture is based on the concept of wrappers (sometimes called transformers) and
mediators. Each wrapper is a piece of code that deals with a particular data source (called info-source), which receive a query expressed in Object Exchange Model Query Language (OEM-QL), transforms the query into a language understandable for the data source, and retrieves the required information using a Object Exchange Model (OEM) (Quass et al., 1995). A mediator is able to deal with other mediators or wrappers in order to make some processing before and after sending information to the application or user (Garcia-Molina et al., 1997). A user (or an application) can interact with a wrapper or a mediator using the OEM-QL for querying.

Wrappers and mediators are able to handle structured and semi-structured data coming from the data source. Mediators or wrappers may use a local proprietary database for improving performance when the amount of information to be returned is large (instead of using principal memory) (Quass et al., 1995).

2. **Infomaster.** Infomaster (Genesereth et al., 1997) is an information integration system that provides integrated access to multiple distributed heterogeneous information sources on the Internet. Infomaster creates a virtual data warehouse. The core of Infomaster is a facilitator that determines which source contains the information necessary to answer the query, designs a strategy for answering the query, and performs translations to convert source information to a common form or the form requested by the user.

Infomaster handles both structural and content translation to resolve differences between multiple data sources and the multiple applications. Infomaster connects to a variety of databases using wrappers. Infomaster uses rules and constraints to describe information sources and translations among these sources. These rules and constraints are stored in a knowledge base.

3. **InfoSleuth.** InfoSleuth (Bayardo et al., 1997) is an agent-based system for information discovery and retrieval in a dynamic and open environment. InfoSleuth has its roots in the
Carnot project (Huhns and Singh, 1997), which specialized in integrating heterogeneous information sources. The InfoSleuth project extends the capabilities of the Carnot technologies into dynamically changing environments.

The InfoSleuth system consists of a set of collaborating agents that work together for information discovery and retrieval in a dynamically changing environment such as the World Wide Web (Bayardo et al., 1997). It focuses on:

- gathering information via complex queries from a changing set of databases and semi-structured text repositories distributed across an internet,
- performing polling and notification for monitoring changes in data, and
- Analyzing gathered information using statistical data mining techniques and/or logical inference.

The InfoSleuth agents are organized as core agents that provide basic information subscription, filtering and fusion capabilities, resource agents that serve as interface to external information sources, and user agents that act as proxies for individual users or groups of users.

### 2.5 Shortcoming of Reviewed Ontology-based Web Data Integration Systems

The reviewed web data integration systems in Section 2.4 utilize ontology technology in order to resolve semantic conflicts. SIMS (Arens et al., 1997), KRAFT (Visser et al., 2000) and MOMIS (Beneventano et al., 2004) create one global or shared ontology and subsequently, they perform semantic mapping between the created global or shared ontology and the local schemas or ontologies. In the web context the maintenance and updating of global or shared ontology and semantic mappings information are very time
consuming and costly because many web data sources are involved and the number of involved web data sources change frequently. COIN (Goh et al., 1999) uses a domain model as a single ontology for reconciliation of semantic conflicts between heterogeneous data sources. Using a single ontology in the web data context is not recommended because communities should be free to use their own vocabularies and semantics according to their requirements. Therefore any web data source can have its own ontology before the creation of any web data information system. OBSERVER (Mena et al., 2000) uses multiple ontologies for reconciliation of semantic conflicts between heterogeneous data sources. The major problem in OBSERVER is in the detection of similarities relations among ontologies and performing semantic mapping. OBSERVER merges ontologies based on existing detected similarity relations kept in system. The merging process of web ontologies in OBSERVER is highly costly with respect to time because the system needs to frequently merge two web ontologies in mapping time (because web ontologies change frequently).

Table 2.1 summarizes the shortcomings of the reviewed projects. In our proposed approach we try to overcome semantic heterogeneities by using a domain specific ontology and we resolve semantic problems between web sources through semantic mapping between the domain ontology and the local ontologies. Our approach for web data integration system architecture is close to the OBSERVER project (Mena et al., 2000).
2.6 Semantic Ontology Mapping Projects

There are many different web ontologies with their underlying data sources in the context of semantic web. For interoperability purposes across heterogeneous web sources we need to integrate these web ontologies semantically. Semantic mapping between web ontologies is a major task in semantic integration process.

<table>
<thead>
<tr>
<th>Project</th>
<th>Ontology architecture</th>
<th>Shortcoming</th>
</tr>
</thead>
<tbody>
<tr>
<td>COIN Project</td>
<td>Single ontology + Mediator</td>
<td>Communities are not free to use their own vocabularies and semantics according to their requirements.</td>
</tr>
<tr>
<td>MOMIS Project</td>
<td>Single ontology By integrating of local schemas</td>
<td>Creating and Updating of global ontology is very time consuming and costly.</td>
</tr>
<tr>
<td>KRAFT Project</td>
<td>Single-Shared ontology map to local ontologies</td>
<td>Creating and maintenance of a single-shared ontology and semantic mappings information are very time consuming and costly.</td>
</tr>
<tr>
<td>OBSERVER Project</td>
<td>Multiple ontology</td>
<td>The lack of a common vocabulary makes it difficult to detect of similarities relations among ontologies and performing semantic mapping.</td>
</tr>
<tr>
<td>SIMS Project</td>
<td>Single ontology</td>
<td>Communities are not free to use their own vocabularies and semantics according to their requirements.</td>
</tr>
</tbody>
</table>
In this section, we survey related works concerned with semantic integration process between local ontologies. Our method (please refer to chapter 5) for semantic mapping between web ontologies is based on the results of these works.

1. GLUE. GLUE is a system that employs machine learning techniques to find semantic mappings between ontologies (Doan, et al., 2003). The architecture of GLUE consists of three main modules: Distribution Estimator, Similarity Estimator, and Relaxation Labeler. The Distribution Estimator takes as input two taxonomies $O_1$ and $O_2$, together with their data instances. Then it applies machine learning techniques to compute for every pair of concepts, their similarity. The Distribution Estimator uses a set of base learners and a meta-learner. Next, the Similarity Estimator computes a value for each pair of concepts that denotes the similarity between the concepts. The output from this module is a similarity matrix between the concepts in the two taxonomies. The Relaxation Labeler module then takes the similarity matrix, together with the domain-specific constraints and heuristic knowledge, and searches for the mapping configuration that best satisfies the domain constraints and the common knowledge. This mapping configuration is the output of GLUE.

GLUE gives a choice to users for several practical similarity measures. GLUE has a total of three learners: Content Learner, Name Learner, and Meta Learner (Choi et al., 2005). Content and Name Learners are two base learners, while Meta Learner combines the two base learners’ prediction. The Content Learner exploits the frequencies of words in content of an instance (concatenation of attributes of an instance). The Name Learner uses the full name of the input instance. The Meta-Learner combines the predictions of base learners and assigns weights to base learners based on how much it trusts that learner’s predictions. In GLUE, Relaxation Labeling takes a similarity matrix and searches for the mapping (best
label assignment between nodes (concepts). This mapping configuration is the output of GLUE.

**2. QOM.** QOM (Quick Ontology Mapping approach) (Ehrig and Staab, 2004) is a semantic mapping algorithm between ontologies. QOM has lower run-time complexity than existing prominent approaches that is order of $n \cdot \log(n)$ (Ehrig and Staab, 2004). QOM performs semantic mapping process in six steps as following:

*Step 1: Feature engineering:* Firstly, the ontologies have to be represented in RDFS (Resource Description Framework Schema).

*Step 2: Search Step Selection:* Entities of the first ontology are compared with entities of the second ontology. QOM uses heuristics to lower the number of candidate mappings. QOM combines the subsequent strategies to propose new candidate mappings for inspection: First, QOM randomly selects a fixed number of candidate mappings from all possible mappings. Second, QOM restricts candidate mappings to entity pairs whose labels are near to each other in a stored list. Third, QOM compares only entities for which adjacent entities were assigned new mappings in a previous iteration.

*Step 3: Similarity Computation:* Determines similarity values between pairs of entities based on their definitions in ontologies. In order to optimize algorithm, QOM has restricted the range of costly features.

*Step 4: Similarity Aggregation:* The different similarity values for one candidate pair must be aggregated into a single aggregated similarity value.

*Step 5: Interpretation:* uses the individual or aggregated similarity values to derive mappings between entities from ontologies.

*Step 6: Iteration:* QOM iterates to find mappings based on lexical knowledge first and based on knowledge structures later. QOM restricts the number of runs to ten.
3. MAFRA. **MAFRA** (Ontology MAppling FRAmework for distributed ontologies in the Semantic Web) (Silva and Rocha, 2003) provides an Ontology Mapping Framework supporting the interactive, incremental and dynamic ontology mapping process in the context of the Semantic Web. The MAFRA framework consists of five horizontal modules and four vertical components. Four vertical modules run along the entire mapping process, interacting with horizontal modules. Five horizontal modules are as follows:

1) **Lift & Normalization**: Both ontologies must be normalized to a uniform representation.

2) **Similarity**: This module finds out and establishes similarities between source ontology entities and target ontology entities.

3) **Semantic Bridging**: It defines procedural mapping information for transforming source instances into the most similar target instances.

4) **Execution**: It transforms instances from the source ontology into target ontology by consulting the semantic bridges.

5) **Post-processing**: It takes the result of the execution module to check and improve the quality of the transformation results.

Four vertical modules are as follows:

1) **Evolution**: It maintains semantic bridges in synchrony with the changes in the source and target ontologies.

2) **Cooperative Consensus Building**: It is responsible for establishing a consensus on semantic bridges between two communities in the mapping process.

3) **Domain Constraints and Background Knowledge**: It improves similarity measure and semantic bridge by background knowledge, e.g. by using domain-specific thesauri.

4) **Graphical User Interface (GUI)**: This module allows the user to interact with the system for refinement of mapping.
The role of the semantic bridging module is to semantically relate entities from the source and target ontologies. Semantic bridges map entities in two different ontologies consisting of concept and property bridges. The concept bridge translates source instances into target ones. The property bridge transforms source instance properties into target instance properties.

4. Chimaera. Chimaera (McGuinness, et al., 2000) is an ontology merging and diagnosis tool developed by the Stanford University Knowledge Systems Laboratory (KSL). Its initial design goal was to provide substantial assistance with the task of merging knowledge bases produced by multiple authors in multiple settings. Later, it took on another goal of supporting testing and diagnosing ontologies as well. Finally, inherent in the goals of supporting merging and diagnosis are requirements for ontology browsing and editing. It is mainly targeted at lightweight ontologies. Its design and implementation are based on other applications such as the Ontolingua ontology development environment (Farquhar et al., 1996). The two major tasks in merging ontologies that Chimaera supports are (1) coalesce two semantically identical terms from different ontologies so that they are referred to by the same name in the resulting ontology, and (2) identify terms that should be related by subsumption, disjointness, or instance relationships and provide support for introducing those relationships. There are many auxiliary tasks inherent in these tasks, such as identifying the locations for editing, performing the edits, identifying when two terms could be identical if they had small modifications such as a further specialization on a value-type constraint, etc. Chimaera generates name resolution lists that help the user in the merging task by suggesting terms from a different ontology that are candidates to be merged or to have taxonomic relationships not yet included in the merged ontology. It also generates a taxonomy resolution list where it suggests taxonomy areas that are candidates for reorganization. It uses a number of heuristic strategies for finding such edit points.
5. Anchor-PROMPT. Anchor-PROMPT (Noy and Musen, 2001) is an algorithm that finds semantically similar terms automatically. The Anchor-PROMPT algorithm was developed for improvement of PROMPT tools (Noy and Musen, 2000) and then development of Prompt Suit tools (Noy and Musen, 2003). PROMPT and PROMPT Suit are tools for semi-automatic guided ontology merging. It is a plugin for Protege (Noy et al., 2000).

Anchor-PROMPT constructs a directed labeled graph representing the ontology from the hierarchy of concepts (called classes in the algorithm) and the hierarchy of relations (called slots in the algorithm), where nodes in the graph are concepts and arcs are relations denoting relationships between concepts (the labels are the names of the relations). An initial list of anchors-pairs of related concepts defined by the users or automatically identified by lexical matching is the input for the algorithm. Anchor-PROMPT analyzes the paths in the sub-graph limited by the anchors and it determines which concepts frequently appear in similar positions on similar paths. Based on these frequencies, the algorithm decides if these concepts are semantically similar concepts. However, Anchor-PROMPT finds only concept mappings, not relation mappings and it uses relation names for labels on the arcs and simple string comparison of these labels. So if the relation names are differently defined, the algorithm will not work well. The returned results of the algorithm will also be limited if the structures of the ontologies are different (e.g. one is deep with many inter-linked concepts, and the other is shallow). The algorithm will have problems if a hierarchy has only a few levels and most of the relations are associated with the concepts at the top of the hierarchy (Choi et al., 2005; Ehrig and Staab, 2004; Noy, 2004).

6. Asco. Asco (Thanh-Le et al., 2004) is an algorithm for matching ontologies. Asco algorithm calculates the similarity of two classes (or two relations) in two ontologies by
using the information about the names, the labels, the descriptions of the classes (or the relations). The similarity of classes (or relations) will be calculated based on the different information (e.g. names, labels…) and the obtained similarity results will be combined to have the final similarity value. ASCO algorithm has two phases: a linguistic phase and a structural phase. In the linguistic phase, the similarity of two classes or relations in two ontologies is calculated by relying on linguistic components of their names, their labels and their descriptions. For each of these three elements, a similarity value is produced. A combination of these values is a linguistic similarity value of the two classes or relations. Asco algorithm exploits WordNet (Miller, 1995) as auxiliary information for finding of synonyms of a term. In structural phase, Asco algorithm relies on two structural matching: adjacency and path; In adjacency, the algorithm relies on the hypothesis that if the direct super-concepts and/or the direct sub-concepts and/or the sibling concepts of two concepts are already similar, the two concepts in question may be also similar. In path matching, the algorithm’s intuition is that if the path from the root of the first hierarchy of concepts to concept A contains similar concepts with the path from the root to concept B in the second hierarchy of concepts, the two concepts A and B may be similar too.

The output of ontology matching is a list of mappings between the elements of the ontologies such as concepts, relations. These mappings are generated by relying on the final similarity value between two elements, which is computed from the linguistic and structural similarity values.
2.7 Comparison of Reviewed Semantic Mapping Approaches

To effectively compare the above studied systems, we need to develop a comparison framework. (Rahm and Bernstein, 2001) and (Noy and Musen, 2002) proposed a set of evaluation criteria for ontology mapping and merging tools, as follows: (we consider each mapping approach consisting of one mapping algorithm and each mapping algorithm consists of some mapping functions)

- Number of mapping function: single or multiple; mapping algorithm uses one or more mapping function for finding similar terms (mapping pair elements).
- Number of mapping criterion: single or multiple; each mapping function uses one or more mapping criteria for comparison of similarity of two elements.
- Combination of mapping functions: Individual or Composition or Hybrid; mapping algorithm employs one mapping function with one mapping criteria (individual) or mapping algorithm employs combination of multiple mapping functions by combining results of multiple mapping function within a composite mapper or by using multiple mapping criteria within an integrated hybrid mapper.
- Mapping granularity: element vs structure level; mapping algorithm finds similarity in element granularity (such as concept, attribute and relationship similarity) or for combinations of elements and complex structures (such as sub-tree, path, sub-graph, table).
- Mapping technique: linguistic-based, constraint-based and structural-based; mapping algorithm finds similarities by a linguistic-based approach such as element-name matching (string matching) or by constraint-based approach (e.g., based on key properties) or structural-based (e.g. based on children, siblings).
• Mapping element: Concept, Instance, Attribute or Relationship; mapping function maps types of ontology terms and elements to each other. The mapping algorithms differ on what kind of elements in the ontology they relate in their output.

• Mapping cardinality: 1:1, 1:n, n:1 or n:m; a mapping algorithm may relate one or more elements of one ontology to one or more elements of the other.

• Ontology representation model: mapping approach uses what data structure model for representation of ontologies.

• Auxiliary information: mapping algorithm may rely not only on the input ontologies but also on auxiliary information, such as dictionaries, user inputs and previous mapping results.

• Level of user interaction. Mapping algorithm might work automatically or interactively. In the latter case, it can build on feedback from the user to improve the quality of the mapping.

• Type of output. The output of the mapping algorithm can be a set of mapping information and rules (defining similarities and differences), a single merged ontology, an instantiated mapping ontology in a particular language, pairs of related concepts, etc.

We studied the characteristics of the proposed approaches using a set of criteria, which combines the evaluation framework in (Rahm and Bernstein, 2001; Noy and Musen, 2002). Figure 2.3 shows what the six studied systems have in common and in difference according to the set of criteria.
<table>
<thead>
<tr>
<th>Mapping function specification</th>
<th>MAFRA</th>
<th>Chimaera</th>
<th>Anchor-PROMPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of mapping function</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>Number of mapping criterion</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>Combination of mapping function</td>
<td>Hybrid</td>
<td>Hybrid</td>
<td>Hybrid</td>
</tr>
<tr>
<td>Mapping algorithm technique</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mapping technique</td>
<td>Name similarity, structural similarity</td>
<td>Name similarity, subsumption, disjointness or instance relationships.</td>
<td>Name similarity, class hierarchy, paths matching.</td>
</tr>
<tr>
<td>Mapping granularity</td>
<td>Element-level</td>
<td>Element-level</td>
<td>Element-level</td>
</tr>
<tr>
<td>Mapping element</td>
<td>Concept, attribute, relation</td>
<td>Concept, relationship</td>
<td>Concept, relationship, constraint</td>
</tr>
<tr>
<td>Mapping cardinality</td>
<td>1:1, 1:n, m:1</td>
<td>1:1</td>
<td>1:n</td>
</tr>
<tr>
<td>Input &amp; output</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ontology representation model</td>
<td>RDFS</td>
<td>OKBC knowledge model</td>
<td>OKBC knowledge model</td>
</tr>
<tr>
<td>Auxiliary information</td>
<td>Domain-specific thesauri, WordNet</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Input of algorithm</td>
<td>Two input ontologies</td>
<td>Two input ontologies</td>
<td>Two input ontologies</td>
</tr>
<tr>
<td>Output of algorithm</td>
<td>Semantic bridging ontology</td>
<td>A merged ontology</td>
<td>A merged ontology</td>
</tr>
<tr>
<td>User interaction</td>
<td>Drive the mapping process, provide domain constraint, create semantic bridges, refine semantic bridge</td>
<td>NO</td>
<td>Approving disapproving or changing the system suggestions</td>
</tr>
</tbody>
</table>

Fig. 2.3 Comparison framework of reviewed mapping approaches
<table>
<thead>
<tr>
<th>Mapping function specification</th>
<th>QOM</th>
<th>Asco</th>
<th>GLUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of mapping function</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>Number of mapping criterion</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>Combination of mapping function</td>
<td>Hybrid</td>
<td>Composition</td>
<td>Composition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping algorithm technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping technique</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping granularity</th>
<th>Element-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping element</td>
<td>Concept, relation, instance</td>
</tr>
<tr>
<td>Mapping cardinality</td>
<td>1:1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input &amp; output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology representation model</td>
</tr>
<tr>
<td>Auxiliary information</td>
</tr>
<tr>
<td>Input of algorithm</td>
</tr>
<tr>
<td>Output of algorithm</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approving disapproving or adjusting the system suggestions</td>
</tr>
</tbody>
</table>

Continue Fig. 2.3
The figure shows all approaches use more than one basic mapping function, either in a hybrid way or in a combined way. Most approaches provide both structural level and element level matching, in particular name and graph structure based match. However, only two of the systems consider instance data and 1:1 match as the main focus. The main forms of output are a merged ontology or a list of pairs of related concepts.

Even though each language choose a particular representation language to base their implementation on, it is possible to map the source ontology to most common types of ontology representation languages. This is also partly because most the studied systems consider only the core elements of ontologies, like concepts and relations, which most of the representation languages support. Furthermore, it is possible to translate between different representational languages (Chalupsky, 2000).

---

2.8 Shortcomings of Reviewed Semantic Ontology Mapping Approaches

The studied semantic ontology mapping approaches in Section 2.6, exploit the information available in ontologies and map similar terms of two given ontologies to each other. We can evaluate an ontology mapping algorithm based on two main factors: the quality of the mapping results and the run time complexity of the mapping approach (Klein, 2001; Mitra and Wiederhold, 2002). The quality of mapping approach is based on how many of the ontology mismatch levels are covered and solved by the mapping algorithm. The reviewed approaches do not cover all levels of ontology mismatches and are different in mapping quality. In our proposed ontology mapping algorithm (please refer to Chapter 5), we consider all ontology mismatch levels and try to resolve them. The performance of mapping approaches is based on the runtime complexity of the mapping algorithms. Some
approaches have higher quality of mapping results but they have high runtime complexity (Mitra and Wiederhold, 2002). We must consider both factors; quality and runtime, in the development of the semantic mapping algorithm.

2.9. Architecture of Ontologies in Data Integration System

In all ontology–based integration approaches, the ontologies are used for the explicit description of the semantics of information sources. But there are different ways of how to employ the ontologies. In general, four different directions can be identified: single ontology approaches, multiple ontologies Approaches, Top-level ontology approaches and Shared vocabulary approaches (Wache et al., 2000).

The following paragraphs give a brief overview of the four main ontology architectures.

2.9.1. Single ontology architecture

A single ontology approach uses one global ontology providing a shared vocabulary for the specification of the semantics (shown in Figure 2.4). All information sources are related to the global ontology. A prominent approach of this kind of ontology integration is SIMS (Arens et al., 1997). The SIMS model includes a hierarchical terminology knowledge base. An independent model of each information source must be described for this system by relating the objects of each source to the global domain model. The relationships clarify the semantics of the source objects and help to find semantically corresponding objects (Wache et al., 2000).
The single ontology approach can be applied to integration problems where all information sources to be integrated provide nearly the same view on a domain. If one information source has a different view on a domain, finding the minimal ontology commitment becomes a difficult task (Gruber, 1995). These disadvantages led to the development of the multiple ontology approach (Wache et al., 2000).

### 2.9.2. Multiple ontologies architecture

In a multiple ontology approach, each information source is described by its own ontology (shown in Figure 2.5). For example, in OBSERVER (Mena et al., 2000) the semantics of an information source is described by a separate ontology. In principle, the “source ontology” can be a combination of several other ontologies but it can not be assumed that the different “source ontologies” share the same vocabulary.
At a first glance, the advantage of a multiple ontology approach seems to be that no common and minimal ontology commitment about the global ontology is needed (Gruber, 1995). Each of source ontology could be developed without considering other sources or their ontologies — no common ontology with the agreement of all sources is needed. This ontology architecture can simplify the change, i.e. modifications in one information source or the addition and removal of sources. But in reality the lack of a common vocabulary makes it difficult to compare different source ontologies. To overcome this problem, an additional representation formalism defining the inter-ontology mapping is provided. The inter-ontology mapping identifies semantically corresponding terms of different source ontologies, e.g. which terms are semantically equal or similar between ontologies. In practice the inter-ontology mapping is very difficult to define, because of the many semantic heterogeneity problems which may occur (Wache et al., 2000).

2.9.3. Top-level ontology approach

To overcome the drawbacks of the single and multiple ontology approach, top-level ontology approaches were developed (shown in Figure 2.6). Similar to the multiple ontology approach, the semantics of each source is described by its own separate ontology. In order to perform a suitable matching between ontologies we need a basic agreement between application ontologies. Such agreement is made by means of higher-level ontologies. An ontology for a large number of communities cannot be complete or highly specialized. It is difficult to reach a consensus within the community or between communities over detailed definitions in the ontology. A community can adopt a higher-level ontology and specialize it by adding its own definitions to it. As a result, a specialized ontology cannot remove any constraints or terms of a higher-level ontology without the agreement of the communities already committed to that ontology. In other words, modification of ontologies need agreement among all communities committed to it.
The top-level ontology contains super class terms of a domain. A source ontology can inherit all the definitions from the top-level ontology.

![Top-level ontology architecture](image)

Fig. 2.6 Top-level ontology architecture

The advantage of a top-level ontology approach is that new sources can easily be added without the need to modify the mappings or the top-level ontology. It also supports the acquisition and evolution of ontologies. The use of a top-level ontology makes the source ontologies comparable and avoids the disadvantages of multiple ontology approaches. The drawback of a top-level ontology approach, however, is that, existing ontologies cannot be reused easily, but has to be re-developed from scratch, because all source ontologies have to refer to the top-level ontology (Wache et al., 2000).

2.9.4. Shared ontology or vocabulary approach

To overcome the drawbacks of the above ontology approaches, a shared vocabulary approach was developed (shown in Figure 2.7). Similar to the multiple ontology approach the semantics of each source is described by its own ontology. However, in order to make the source ontologies comparable to each other they are built upon one global shared vocabulary (Goh, 1997; Wache et al., 1999). The shared vocabulary contains basic terms
(the primitives) of a domain. In order to build complex terms of the source ontologies the primitive ones are combined by some operators. Since each term of the source ontology is based on the primitives, the terms become easier compared to the multiple ontology approaches (Stuckenschmidt et al., 2000; Wache et al., 2000).

In the shared vocabulary approach the interesting point is how the local ontologies are described, i.e. how the terms of the source ontology are described by the primitives of the shared vocabulary. In MECOTA (Wache et al., 1999), each source information is annotated by a label which indicates the semantics of the information. The label combines the primitive terms from the shared vocabulary.

The advantage of a shared vocabulary approach is that new sources can easily be added without the need to modify the mappings or the shared vocabulary. It also supports the acquisition and evolution of ontologies. The use of a shared vocabulary makes the source ontologies comparable and avoids the disadvantages of the multiple ontology approach.

The drawback of a shared vocabulary approach however, is that existing ontologies cannot be reused easily, and has to be re-developed from scratch, because all source ontologies have to refer to the shared vocabulary (Wache et al., 2000).
2.9.5. Comparison of ontology architectures

Table 2.2 summarizes the benefits and drawbacks of the different ontology mapping approaches. It’s impossible to use one global ontology, shared vocabulary or common top-level ontology for web sources because there are many web sources on the internet and these sources are independent from each other. Web designers and users are free to use their own terms and vocabulary. The schemata are subject to frequent changes (such as DTDs in XML data). Many data sources are involved and the number of involved data sources change frequently on the Internet. Therefore the multiple ontologies architecture is the most appropriate approach for ontology-based Integration of web sources on the internet because each source ontology could be developed without considering other sources or their ontologies — no common ontology with the agreement of all sources is required. This ontology architecture can simplify the change, i.e. modifications in one information source or the addition and removal of sources. However, in reality the lack of a common vocabulary makes it extremely difficult to compare different source ontologies (Wache et al., 2000).
### Table 2.2 Benefits and drawbacks of the different ontology-based Integration approaches from (Wache et al., 2000)

<table>
<thead>
<tr>
<th></th>
<th>Single ontology architecture</th>
<th>Multiple ontologies architecture</th>
<th>Top-level ontology approach</th>
<th>Shared vocabulary approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Implementation effort</strong></td>
<td>Straight forward</td>
<td>Costly</td>
<td>Reasonable</td>
<td>Reasonable</td>
</tr>
<tr>
<td><strong>Semantic heterogeneity</strong></td>
<td>Similar view of domain</td>
<td>Support heterogeneous views</td>
<td>Support heterogeneous views</td>
<td>Support heterogeneous view</td>
</tr>
<tr>
<td><strong>Adding/removing of sources</strong></td>
<td>Need for some adoption in global ontology</td>
<td>Providing a new source ontology, relating to other ontologies</td>
<td>Providing a new source ontology</td>
<td>Providing a new source ontology</td>
</tr>
<tr>
<td><strong>Comparing of multiple ontologies</strong></td>
<td>Difficult, because of the lack of a common vocabulary</td>
<td>Simple, because ontologies inherit from a top-ontology</td>
<td>Easier, because ontologism use a common ontology</td>
<td></td>
</tr>
</tbody>
</table>

### 2.10 Comparison of Reviewed Web Data integration Systems

Before we present a system architecture for web data integration, we compare the various system architectures of ontology-based web data integration systems. The following table (refer to Table 2.3) summarizes the system architecture specifications of the studied systems:
Table 2.3 Comparison of ontology-based web data integration systems

<table>
<thead>
<tr>
<th>Project</th>
<th>Data integration approach</th>
<th>Semantic schema integration approach</th>
<th>Ontology architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>COIN Project</td>
<td>Virtual approach</td>
<td>On the fly integration</td>
<td>Single ontology + Mediator</td>
</tr>
<tr>
<td>MOMIS Project</td>
<td>Virtual approach</td>
<td>Global schema generation</td>
<td>Single ontology By integrating of local schemas</td>
</tr>
<tr>
<td>KRAFT Project</td>
<td>Virtual approach</td>
<td>On the fly integration</td>
<td>Single-Shared ontology map to local ontologies</td>
</tr>
<tr>
<td>OBSERVER Project</td>
<td>Virtual approach</td>
<td>On the fly integration</td>
<td>Multiple ontology</td>
</tr>
<tr>
<td>SIMS Project</td>
<td>Virtual approach</td>
<td>Global ontology generation</td>
<td>Single ontology</td>
</tr>
</tbody>
</table>

The table shows all the studied systems use the virtual approach for data integration and they are different in semantic schema integration and ontology architecture approach. In developing a web data integration system; we must consider the natural specifications of the web environment. The main natural specifications that affect the development of web data integration systems are as follows:

- There are many web data sources on the internet;
- These web data sources are independent from each other;
- Web designers and users are free to use their own terms and vocabulary;
- The schemata are subject to frequent changes (such as DTDs in XML data);
• The number of involved data sources changes frequently on the Internet;

With regards to these specifications, the proper architectures for web data integration are as follows (we will follow these proper architectures in the development of our system):

• *Virtual approach for data integration:* The materialization approach is not suitable for highly dynamic, open and distributed environment (such as the internet) because this approach is suitable when the schemata are not subject to frequent changes (Hakimpour, 2003). In a frequently changing and dynamic environment we are obliged to frequently modify the centralized repository and this is high costly. Therefore, we follow the virtual approach for the design of our system architecture.

• *On the fly integration for semantic schema integration:* The integration of schemata is not beneficial due to the frequent changes of web data. Therefore, queries are translated for the component data sources and the response to the query, in turn, is translated on-the-fly (Hakimpour, 2003).

• *Multiple ontology for ontologies architecture:* The multiple ontologies architecture is the most appropriate approach for ontology-based integration of web sources on the internet because each source ontology could be developed without considering other sources or their ontologies. This ontology architecture can simplify the change, i.e. modifications in one information source or the addition and removal of sources. However, in reality the lack of a common vocabulary makes it difficult to compare different source ontologies (Wache et al., 2000). In our proposed approach, we use a domain based ontology on top of multiple ontologies in order to resolve this weakness.

The Table 2.3 shows The OBSERVER approach follows all the above proper architectures. Our approach for web data integration system architecture is close to the OBSERVER project but has two main differences as compared to the OBSERVER
approach in order to cover the main drawbacks of OBSERVER. We will discuss about these differences and explain our proposed approach in Chapter 4 and 5.

2.11 Summary

In this chapter we describe popular approaches for data integration system and semantic schema integration. The goal of this research is to resolve semantic schema heterogeneity conflicts in web data integration process, therefore this chapter focuses on works relating to following issues:

1. Integration of web data sources.
2. Reconciling of semantic heterogeneities between web data sources.
3. Using ontologies to solve heterogeneity problems.
4. Integration of ontologies.
5. Semantic mapping between ontologies.

The projects introduced in Section 2.4 are major research works directly related to issues 1, 2 and 3. In this research, we propose a web data integration system (please refer to chapter 4) as a solution to the limitations of the reviewed projects. The projects introduced in Section 2.6 are research works related to issues 4 and 5. We propose an ontology semantic mapping algorithm (please refer to chapter 5) as solution to these issues.

We reviewed the existing shortcomings among various web data integration systems and semantic ontology mapping algorithms in Sections 2.5 and 2.7. We will focus on these shortcomings and cover these existing gaps in our proposed approaches.

We discussed about different ontology architectures. With regards to specifications of web environment and reviewed approaches, we concluded the proper approaches and
architectures for web data integration as follows (we will follow these proper architectures in our proposed system):

- **Virtual approach for data integration:** In a frequently changing and dynamic environment we are obliged to frequently modify the centralized repository and this is high costly. Therefore, we follow the virtual approach for the design of our system architecture.

- **On the fly integration for semantic schema integration:** The integration of schemata is not beneficial due to the frequent changes of web data. Therefore, queries are translated for the component data sources and the response to the query, in turn, is translated on-the-fly (Hakimpour, 2003).

- **Multiple ontology for ontologies architecture:** The multiple ontologies architecture is the most appropriate approach for ontology-based integration of web sources on the internet because each source ontology could be developed without considering other sources or their ontologies. (Wache et al., 2000).
CHAPTER 3:
SEMANTICS AND
ONTOLOGY
Chapter 3  Semantics and Ontology

3.1 Introduction

Data in information systems and other data sources can now be accessed via the internet. Information systems on the web are modeled and implemented differently according to the requirement of their users. The access, retrieval and utilizing of web data from multiple sources on different information systems require some kind of information integration. Semantic heterogeneity is a major obstacle to perform data integration which refers to misinterpretations of data. Semantic heterogeneity has been identified as one of the most crucial problems when dealing with interoperability and cooperation among multiple data sources on the internet (Kashyap and Sheth, 1998; Decker et al., 1999; Fensel, 2001a).

The goal of this thesis is in resolving semantic heterogeneities of schema level in web data integration systems. In this chapter we clarify and explain concepts concerning semantic integration such as:

- Heterogeneity levels in data integration (discussed in Section 3.2);
- The meaning of the term of semantic (discussed in Section 3.2.1);
- Semantic of schema data and related heterogeneities (discussed in Section 3.2.2);
- Semantic web (discussed in Section 3.3);
- Ontology and its issues and roles (discussed in section 3.4);
- Ontology language (discussed in section 3.5 and 3.6);
3.2 Schema Data Heterogeneity

Integration of heterogeneous data in the internet is a complex activity that involves reconciliation of various levels of conflicts. Before we can integrate the heterogeneous data we need to resolve heterogeneity conflicts. As mentioned in Chapter 1 heterogeneity conflicts are classified in three abstraction levels; data value conflicts, schema conflicts and data model conflicts and conflicts in each level can be categorized into two categories: syntactic and semantic conflicts (please referred to Chapter1: figure 1.5). In the next section, semantic category of schema level, will be thoroughly investigated and studied. The aim of this work is to give a solution for resolving semantic schema heterogeneities in web data integration systems.

3.2.1 What is semantic

In the field of information system technology, some consider semantic and schematic heterogeneity to be same (Bishr, 1997; Garcia-Solaco et al., 1996), while some consider it as only part of schema heterogeneity (Kim et al., 1993). Consequently, approaches to resolve semantic heterogeneity depend on how the term semantics has been interpreted (Hakimpour, 2003). In this section, we describe interpretation of semantic concept in our research.

In the domain of database technology, semantics refers to people’s interpretation of data stored in a database. As stated in (Sowa, 2002):

“The common aspect that unifies all the groups [of specialists] is knowledge of the meaning of the data and the constraints necessary to keep it a faithful model of the real world. The study of the meaning and constraints on the data is called database semantics.”
As defined in (Woods, 1975) “semantics is meaning and the use of data”. In the information systems context, semantics can be viewed as a mapping between an object modeled, represented and/or stored in an information system (e.g., an "object" in a database) and the real-world object(s) it represents. This mapping represents the semantics of the modeled object by describing or identifying the meaning and the use perspectives (Sheth, 1998). As defined in (Rosenthal and Sciore, 1995) data semantics as a connection from a database to the real-world outside the database. As defined in (Meersman, 1994) semantics to be “a (set of) mapping(s) from your representation language to agreed concepts (objects, relationships, behavior) in the real-world”.

There are two beliefs about semantic (Sheth, 1998):

- there is no semantics without some form of (formal, informal) agreement between the agents observing the real-world.
- Semantics always exists but we need an interpretation agent to determine/derive the meaning (associated with an object), and consequently any semantics implies the existence of an agent (interpreter) interacting with a domain.

Therefore, semantics is derived from an agreement between cognitive agents observing the real-world. Semantics always exists, but we need interpretation agents for deriving the meaning (i.e., an existence of outside agents is needed) and that it is useless to talk about semantics without an agreement.

**3.2.2 Semantic heterogeneity in schema level**

We describe data schema through a formal language (Data Definition Language) in the database. Any DDL (Data Definition Language) has one grammar. The grammar is used to validate sentences of a DDL. Grammar is viewed as a set of rules for building valid sentences. In other words, a language is the set of all sentences that conform to its grammar.
Grammar has two main components: symbols and order (Mosses, 1990). Order is represented by relations among symbols. The meaning and interpretation of symbols is the semantic of schema data. Differences in interpretation of the symbols in a database are the source of semantic problems during communication (Mosses, 1990).

There are two types of symbol: terminal and non-terminal (shown in Figure 3.1), as defined in (Mosses, 1990). Terminal symbols (nodes shown with ovals in Figure 3.2) appear in the sentences of a language. For instance, consider the statement in Figure 3.3, All the terms (or symbols) in the sentence of Figure 3.3 are terminal symbols. Non-terminal symbols (nodes shown as rectangles in Figure 3.2) do not appear in the sentences of a language and have their own definition by another symbol or an order of a set of symbols — e.g., “class-name” or “AttributeDef” (shown in Figure 3.2).

Figure 3.1 Classification of symbols or terms in grammar from (Hakimpour, 2003)
There are two kinds terminal symbol: Constant and Identifier. All nodes shown with ovals in Figure 3.2 and Bold in Figure 3.3 (are terminal symbol) appear both in the sentences of the language and in the grammar with no changes and are called constant symbols. Nodes shown as rectangles in Figure 3.2 (are non-terminal symbol) will not appear in the sentences of a language and are replaced by terminal symbols (such as “class name”, “attribute name”, “properties” and “AttributeDef”). The terminal symbols that are used to replace these non-terminal symbols are called identifiers (student, student code, name and address in Figure 3.3). Replacement of non-terminal symbols with the identifiers is done according to the user’s preference (Mosses, 1990; Hakimpour, 2003).

Differences in the interpretation of identifiers lead to semantic schema heterogeneity problem (Hakimpour, 2003). Users of a language (i.e., database designers or programmers)
are free to select their own terms (or symbols) for naming classes, attributes and so on, with no obligation to formally define their intention. The explicit formalization of the interpretation, instead of having to rely on common sense to interpret identifiers’ names, can help to deal with the semantic heterogeneity problem (Hakimpour, 2003). Interpretation of identifier symbols in schemata is the main concern of our thesis.

3.3. Semantic Web

As Berners-Lee has mentioned (Berners-Lee, 1998), the current web is an ocean of information created by organizations, individuals and communities. All the information is linked to each other and users can access them by URI addresses. The current web is actually the web of URI/Links and it is for human consumption. Semantic Web is the web of meaning. We need only make information meaningfully processable by computers. Semantic Web is aimed at computers to produce machine understandable information. Semantic Web represents a set of semantically inter-linked data units and it is created as an information space in which the information is machine readable and processable. The current web is a set of informally inter-linked data units and is an information space, which is aimed at human consumption (Berners-Lee, 1998; Palmer, 2001). Both webs use links to access the information. The current web makes it possible to navigate by using these links. Semantic Web makes it possible to navigate as well share concepts. Semantic links between different concepts can be made. In this way, the machine-readable language is defined (Palmer, 2001; Berners-Lee, 2001).

The human language expresses meaning by referring to a set of shared concepts. This set of concepts makes it possible to understand their communication. Semantic Web works in
the same way. Semantic Web makes it possible for machines to use a set of shared concepts, identified by URIs (Berners-Lee, 2001).

Figure 3.4 illustrates the encoding and decoding of the information on the current web and the semantic Web. The current web does not have any metadata, which can define the meaning of data concepts and explain relationships between concepts. We as a human being, take advantage of our reasoning ability and the concepts that we have defined through other concepts in order to distinguish, for example a research paper from a recipe. The difference between a research paper and a recipe is understood by a human being but not computers as they can not make a wild guess. Semantic Web, as shown in Figure 3.4, makes it possible for computers to understand differences by a set of shared concepts identified by URIs (Pouyan, 2003).

Fig. 3.4 Encoding and decoding of the information on the web (the first figure) and the semantic web (the second figure) from (Pouyan, 2003)
Figure 3.5 shows the architecture of the semantic web. This architectural description is language oriented and has a layered approach. It consists of a set of layers, which represent different concepts.

This layered structure of the semantic web is not the definitive model of the semantic web, but is intended to be a prototype, an idealized diagram. It is designed in such way that “each layer gives progressively more value” (Dumbill, 2001). The architecture of the semantic web starts with the foundation of URIs and Unicode that provide the following layers with an alphabet (i.e. Unicode) and a technology for identifying resources on the web (i.e. URI). The XML layer adds the syntactic interoperability and introduces the RDF + RDF schema layer as the data interoperability layer. The ontology layer deals with the description of objects and the relations among them and is the key layer of the attempt to achieve a shared meaning of a domain of interest. The logic layer provides languages for enabling reasoning on data as they are structured in the lower levels, while the proof layer offers techniques for describing the steps taken for reaching a conclusion from the facts. On the top of the architecture lies the trust layer that provides the means of weighing the value of information, of deductions made and so on. Each of these layers is in interaction.
with the other layers. The lower layer provides primitives and the semantics for the upper layer in order to create a more powerful language (Dumbill, 2001).

3.4. **Ontology**

In the domain of philosophy, ontology is a branch of metaphysics that explains the nature and essential properties of beings or things. It may concern questions such as “What exists?” and “What general sorts of things are there?” From the perspective of cognitive science every branch of science has its own ontology. An ontology is a collection of categories of things that can or do exist in a domain and relations among categories (Sowa, 2002).

In the domain of artificial intelligence ontology is defined as explicit specification of conceptualization (Gruber, 1993). In this domain, ontology has been used for knowledge representation and sharing or reusing knowledge between agents, with emphasis on formalizing the specification of concepts and relations used by the agents. The main motivation behind ontologies is that they allow sharing and reuse of bodies of knowledge in a computational form (Fensel, 2001a). Ontologies also attract attention in the integration of information systems and databases (Guarino, 1998).

3.4.1 **What is ontology**

The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an ontology, is a catalog of the types of things that are assumed to exist in a domain (Sowa, 2002). The word of ontology comes from the Greek *ontos* (being) and *logos* (word). It has been introduced in philosophy in the 19th century, by German philosophers, to distinguish the study of being as such from the study of various kinds of beings in the natural sciences (Guarino, 1998).
Ontologies provide a machine process-able semantics of information sources that can be communicated between different agents (software and humans). The commonly used or highly cited ontology definition is from (Gruber, 1993):

“an ontology is a formal, explicit specification of a shared conceptualization. ‘Conceptualization’ refers to an abstract model of phenomena in the world by having identified the relevant concepts of those phenomena. ‘Explicit’ means that the type of concepts used, and the constraints on their use are explicitly defined. ‘Formal’ refers to the fact that the ontology should be machine readable. ‘Shared’ reflects that ontology should capture consensual knowledge accepted by the communities”.

We consider the following example from (Wang and Kim, 2002), for clarification of ontology concept which is a sub-ontologies of the KA ontology developed by University of Karlsruhe. The Person-ontology defines the types of persons working in academic environments. This ontology defines 10 classes and 23 properties. The overview does not show the relations between classes and properties. Indentation denotes the is-a relationship.

Class hierarchy (10 classes defined):

Person
  Employee
    Academic-Staff
      Lecturer
      Researcher
    Administrative-Staff
      Secretary
      Technical-Staff
  Student
    Ph.D-Student

Properties (relationships and attributes):
Ontology consists of the logical axioms that express the meaning of terms for a particular community. The logical axioms in ontologies define explicit specifications of conceptualization (Gruber, 1993). Logical axioms are the means to introduce concepts, properties (relationships and attributes) and their taxonomic hierarchies and also to express constraints (Hakimpour, 2003). As an example, the following logical axioms for definition of “bus” concept from:

Bus if and only if,

• Bus carry people, and
• Bus has wheels, and
• Bus has driver.

In fact, all the new terms (such as “wheel” or “driver”) introduced in the definition have to be defined as well. One can assign terms such as “Bus” or “Car” to this set logical axioms. Mapping from used vocabulary of a community to logical axioms is the commitment of community to the definitions in the ontology (Guarino, 1998). An ontology exists only under a consensus amongst the members of a community (Bishr et al., 1999; OGC, 1999) e.g., users of one information system or people in one discipline. A community is defined as a set of people who agree upon and commit to the ontology. One can also define a community as set of users in an application domain. However, such a definition will relate ontologies to application domains (Hakimpour, 2003).

The following structure for ontologies was proposed in (Maedche, 2002): an ontology can be described by a 5-tuple consisting of the core elements of an ontology, i.e., concepts, properties hierarchy, a function that relates concepts non-taxonomically and a set of axioms. The elements are defined as follows:
$O: = \{C, P, H, Rel, A\}$ consisting of:

- Two disjoint sets, $C$ (concepts) and $P$ (property: relationship & attribute);
- A concept hierarchy, $H$: $H$ is a directed relation $H \leq C \times C$ which is called concept hierarchy or taxonomy. $H(C1, C2)$ means $C1$ is a sub concept of $C2$;
- A function $Rel: R \rightarrow C \times C$ that relates the concepts non taxonomically. Functions show relationships between concepts.
- A set of ontology axioms $A$, expressed in appropriate logical language. Axioms are a formal description of concepts and properties.

Finally, it is important to state the difference between a thesaurus and an ontology. A thesaurus applies a limited number of known relations among terms (such as synonym and antonym) in contrast; an ontology describes the relationships among the concepts referred by terms. Ontologies define concepts and their relationships (Hakimpour, 2003).

### 3.4.2. Role of ontology

Applying ontologies guarantees consistency in communities’ understanding of statements made during a communication. However, they will not necessarily represent the complete conceptualization of every community committed to the ontology. By using ontologies, communities are able to communicate based on their defined ontologies and the complication of building ontologies is the cost of resolving ambiguity and semantics heterogeneity in communication (Hakimpour, 2003).

An advantage of applying ontologies is that they help applications to be independent of the implicit background knowledge of the community or at least reduce the dependency on such knowledge. One has to explicitly say what his or her intention is by referring to a term. This reduces the chance of semantic heterogeneity in communications and amongst the communities —i.e., their respective information systems (Hakimpour, 2003). Using
ontologies gives the communities the freedom to define their conceptualization — rather than forcing ontology on a community. By using ontologies communities are supposed to communicate through their defined ontologies and the imposed complication of building the ontologies is the cost for resolving ambiguity and semantic heterogeneity (Hakimpour, 2003).

### 3.4.3. Application area of ontology

Currently computers are turning into a worldwide network of information exchange and business transactions. Therefore, support in the exchange of data, information, and knowledge is becoming the key issue in computer technology today. Ontologies provide a shared and common understanding of a domain that can be communicated between people and across application systems. Ontologies will play a major role in supporting information exchange processes in various areas (Fensel, 2001a; Fensel, 2001b). Ontologies were first used in Artificial Intelligence to facilitate knowledge sharing and reuse. More recently, the notion of ontology is also becoming widespread in fields such as intelligent information integration, cooperative information systems, information retrieval, electronic commerce, and knowledge management. The reason ontologies are becoming so popular is due to what they promise: a shared and common understanding of some domain that can be communicated between people and application systems (Fensel, 2001a). Currently, we see ontologies applied to the World Wide Web creating what is called the semantic web (Berners-Lee, 2001).

The three main application areas of ontology technology are Knowledge Management, Web Commerce, and Electronic Business. We briefly discuss these application areas by referring to (Fensel et al., 2001):

**Knowledge Management** is concerned with acquiring, maintaining, and accessing knowledge of an organization. Due to globalization of the Internet, many organizations are
geographically dispersed. With the large number of on-line documents, several document management systems have entered the market. However these systems have the following weaknesses:

- **Searching information**: Existing keyword-based search retrieves irrelevant information.

- **Extracting information**: Human browsing and reading is currently required to extract relevant information from information sources. Automatic agents lack common sense knowledge required to extract such information from textual representations, and they fail to integrate information spread over different sources.

- **Maintaining** weakly structured text sources is a difficult activity. Keeping such collections consistent, correct, and up-to-date requires a mechanized representation of semantics and constraints that help to detect anomalies.

Using Ontologies, semantic annotations will allow structural and semantic definitions of documents providing completely new possibilities: Intelligent search instead of keyword matching, query answering instead of information retrieval, document exchange between departments via ontology mappings (Fensel et al., 2001).

**Web Commerce (B2C)**: Electronic Commerce is becoming an important business area. This is happening for two reasons. First, electronic commerce is extending existing business models. It reduces cost and extends existing distribution channels and may even introduce new distribution possibilities. Second, it enables completely new business models or gives them a much greater importance than they had before. Examples of business field extensions are on-line stores; examples of new business fields are shopping agents and on-line marketplaces that make comparison shopping or meditation of shopping processes into a business. The advantages of on-line stores and the success story of many
of them have led to a large number of such shopping pages. The new task for a customer is now to find a shop that sells the product he is looking for, getting it in the desired quality, quantity, and time, and paying as little as possible for it. Achieving these goals via browsing requires significant time and will only cover a small share of the actual offers. Very early, shop agents were developed that visit several stores, extract product information and present to the customer an instant market overview. Their functionality is provided via wrappers that need to be written for each on-line store. Such a wrapper uses a keyword search for finding the product information together with assumptions on regularities in the presentation format of stores and text extraction heuristics. This technology has two severe limitations:

- **Effort**: Writing a wrapper for each on-line store is a time-consuming activity and changes in the outfit of stores may cause high maintenance efforts.
- **Quality**: The extracted product information is limited (mostly price information), error prone and incomplete. For example, a wrapper may extract the direct product price but misses indirect costs such as shipping costs etc.

These problems are caused by the fact that most product information is provided in the natural language, and automatic text recognition is still a research area with significant unsolved problems. However, the situation may change in the near future when standard representation formalisms for the structure and semantics of data are available. Software agents then can *understand* the product information. Meta-on-line stores can be built with little effort and this technique will also enable complete market transparency.

The low-level programming of wrappers based on text extraction and format heuristics can be replaced by ontology mappings, which translate different product descriptions into each
other. Ontology describes the various products and can be used to navigate and search automatically for the required information (Fensel et al., 2001).

**Electronic Business (B2B):** Electronic Commerce in the business to business field (B2B) is not a new phenomenon. In order to exchange business transactions sender and receiver have to agree on a common standard (a protocol for transmitting the content and a language for describing the content). A number of standards arose for this purpose. One of them is the UN initiative *Electronic Data Interchange for Administration, Commerce, and Transport (EDIFACT)*. The exchange of business data via extranets is not integrated with other document exchange processes, i.e., EDIFACT is an isolated standard. Using the infrastructure of the Internet for business exchange will significantly improve this situation. Standard browsers can be used to render business transactions and these transactions can be transparently integrated into other document exchange processes in the intranet and internet environments. However, this is currently hampered by the fact that HTML does not provide a means for presenting rich syntax and semantics of data. XML, which is designed to close this gap in current Internet technology, will therefore drastically change the situation. B2B communication and data exchange can then be modeled with the same means that are available for the other data exchange processes, transaction specifications can easily be rendered by standard browsers, maintenance will be cheap. XML will provide a standard serialized syntax for defining the structure and semantics of data. Still, it does not provide standard data structures and terminologies to describe business processes and exchanged products. Therefore, ontologies will have to play important roles in XML-based electronic commerce (Fensel et al., 2001):

- *Standard ontologies* have to be developed covering the various business areas. In addition to official standards, if they can attract significant shares of the on-line
transactions in a business field they will factually create a standard ontology for this area. Examples are: Dublin, Common Business Library (CBL), Commerce XML (cXML), Open Applications Group Integration Specification (OAGIS), Open Catalog Format (OCF), Open Financial Exchange (OFX), Real Estate Transaction Markup Language (RETML).

- **Ontology-based translation services** between different data structures in areas where standard ontologies do not exist or where a particular client wants to use his own terminology and needs translation service from his terminology into the standard. This translation service must cover structural and semantic as well as language differences.

Then, ontology-based trading will significantly extend the degree to which data exchange is automated and will create complete new business models in the participating market segments (Fensel et al., 2001).

---

### 3.5 Ontology Language

An ontology defines the terms which is used to describe and represent an area of knowledge. Ontologies are used by people, databases, and applications that need to share domain information (a domain is just a specific subject area or area of knowledge, like medicine, tool manufacturing, real estate, automobile repair, financial management, etc.).

Ontologies are usually expressed in a logic-based language in order to obtain detail, accurate, consistent and meaningful description of the concepts, properties, and relations in
a domain. (Antonion and Harmelen, 2003) states some of the essential requirements for ontology languages as following:

1. **A well-defined syntax**: Ontology language should have a well-defined syntax; it is a necessary condition for machine-processing of information.

2. **A well-defined semantics**: Formal semantics describes precisely the meaning of knowledge. One use of formal semantics is to allow humans to reason about the knowledge. For example:
   - Class membership: If X is an instance of a class C, and C is a subclass of D, then we can infer that X is an instance of D.
   - Equivalence of classes: If class A is equivalent to class B, and class B equivalent to class C, then A is equivalent to C, too.

3. **Efficient reasoning support**: Semantics is a prerequisite for reasoning support. Derivations such as the above (mentioned in well-defined semantic part) can be made by machine, instead of being made by hand.

4. **Sufficient expressive power**: Expressive power of formal semantics causes automated reasoning support allows one to check many more reasoning cases than what can be done manually. This feature helps to:
   - Designing large ontologies, where multiple authors are involved;
   - Integrating and sharing ontologies from various sources.

A language to support the representation of ontologies should provide a set of suitable formalism mechanisms. We discuss these formalism mechanisms in the next section.
3.5.1 Formalism mechanisms

We need to have enough formalism mechanisms to support the representation of ontologies. (Hakimpour, 2003) states the essential mechanisms for semantic definition of concepts (formalization of logical axioms) in ontology as the following:

a. **Concept definition:** Concept definition is a means to define intentional relations (logical axioms) of arity one. A concept is defined by a set of constraints that an instance should satisfy to belong to that concept. In concept definitions we need sub-concept (is-a-type-of) relations for establishing a taxonomy hierarchy of concepts. Concept definitions in ontology are concerned with properties that distinguish its instances from other concepts. As an example, the following logical axioms for definition of “Bus” concept:

Bus(x) if and only if,

- X carry people, and
- X has wheel, and
- X has driver.

b. **Relationship definition:** Relationship definition is a means to define intentional relations of arity two or higher. Relationships in ontologies may be defined independent of concept definitions. For example definition of spouse relationship as: “someone gets married with someone else”. This gives an identity to relationships independent of concept definitions as man or woman. This facilitates support for the specialization of relationships.

Another mechanism offered by some systems is called a Role. A Role is a mechanism that allows us to define or redefine a relationship in a concept definition. For example, while a relationship spouse maps persons to persons, the role spouse defined for concept “man” can have a constraint specifying that the spouse of a (heterosexual) man should be woman; and the same role may be defined for the concept “woman” with the opposite constraint.
Note that in practice, one can specialize relationships according to their domain by a more complicated relationship definition without using the role mechanism. That is, one can express such constraints in the definition of the spouse relationship instead of specializing it in definition of the concept “man”. For example definition of spouse relationship as: “man gets married with woman” and “woman gets married with man” (Hakimpour, 2003).

c. **Instance definition:** Instances represent members of the domain. They are defined with respect to concepts or relationships by is-instance-of relations. Every instance of a concept has a collection of facts. Facts related to every instance of a concept are represented by means of their relations (or roles). For example, an instance of the concept person can be defined by its roles such as its social security number, its name and/or its brother. An instance of a concept can play a role for another instance. As an example, a person can play the role of owner for a car or parent for another person. The two persons in the last example take part in an instance of a parenthood relation. That means, relations have instances as well as concepts. This instance of relation parenthood can be classified as a motherhood or fatherhood (Hakimpour, 2003).

d. **Constraints (or restrictions) definition:** Constraints are conditional phrases mainly used for classifying concepts, relations or individuals (instances). Reasoning systems are able to classify instances under concept or relation definitions by using constraints. As an example, a motherhood relation is parenthood with a constraint on its range that it should be a woman; or a woman is a person with the constraint that its gender should be female.

As far as language can provide the above-mentioned features it can be used to formalize ontologies. In the next section we focus on OWL- newly approved standard for formalizing of web ontologies and briefly discuss about OWL formalism mechanisms (Hakimpour, 2003).
3.6 OWL (Ontology Web Language)

RDFS (RDF Schema) is recognizable as an ontology language but RDFS is not a suitable foundation for ontology representation. It is too weak to describe resources in sufficient detail. Requirements for web ontology language are as follows (Berners-Lee, 1998; Berners-Lee, 2001):

- Compatible with existing Web standards (XML, RDF, RDFS)
- Easy to understand and use
- Adequate expressive power
- Possible to provide automated reasoning support

Two languages have been developed to satisfy the above requirements, OIL (Ontology Inference Layer) developed by group of (largely) European researchers and DAML-ONT developed in DARPA DAML program. Afterwards, these two efforts were merged and produced DAML+OIL. W3C uses DAML+OIL as basis for standardization and developing the OWL language standard. OWL “layered” on top of RDFS and has RDFS based syntax and ontological primitives (subclass etc.) with additional formalism mechanisms (Antonion and Harmelen, 2003).

OWL became a W3C (World Wide Web Consortium) Recommendation in February 2004. Acceptance of a standard encouraged researchers to propose algorithms that rely more heavily on features of the ontology language to compare ontologies.

OWL was designed to provide a common way to process the content of web information (instead of displaying it). OWL was designed to be read by computer applications (instead of humans). OWL can be used to explicitly represent the meaning of terms in vocabularies.
and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web (Berners-Lee, 2001; Dean et al., 2003; McGuinness and Harmelen, 2003).

OWL is written in XML. By using XML, OWL elements (entities) can easily be exchanged between different types of computers using different types of operating system and application languages. OWL can interoperate with RDF models and they are much of the same thing, but OWL is a stronger language with greater machine interpretability than RDF. OWL comes with a larger vocabulary and stronger syntax than RDF. OWL is part of the growing stack of W3C recommendations related to the Semantic Web (McGuinness and Harmelen, 2003) (shown in Figure 3.5).

OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.

RDF is a datamodel for objects ("resources") and relations between them provides a simple semantics for this datamodel.

XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.

XML provides a surface syntax for structured documents.

Fig. 3.5 Stack of W3C recommendations related to the Semantic Web
We use OWL for formalizing domain ontologies in implementation of our proposed approach in this thesis. In the next section, we explain briefly about concept and relation definition mechanisms in OWL.

### 3.6.1 Definition mechanisms in OWL

We categorize OWL definition mechanisms to four categories as follows:

1. Concept definition mechanisms
2. Relation definition mechanisms
3. Instance definition mechanisms
4. Constraint definition mechanisms

The explanation of all OWL mechanisms in detail is not in the scope of this thesis and we refer the reader to (Dean et al., 2003; Smith et al., 2003; Patel-Schneider et al., 2004). For the purpose of this, in this section we briefly explain the OWL mechanisms for the definition of concept elements and relation elements of an ontology and then mention OWL mechanisms for instance and constraint definition. We used (Dean et al., 2003; Smith et al., 2003; Patel-Schneider et al., 2004) from w3c.org as reference for this section.

**Concept definition mechanisms:** OWL uses the term “class” as a substitute for the term “concept”. In OWL, A class description (concept definition) is the basic building blocks of class axioms (informally called class definitions). OWL contains nine mechanisms for class description as following (Dean et al., 2003; Smith et al., 2003; Patel-Schneider et al., 2004):

1. **A class identifier:** It describes a class through a class name (syntactically represented as a URI reference).
2. **An enumeration of individuals:** This type class description describes a class that contains exactly the enumerated individuals. This enables a class to be described by enumerating its instances.

3. **The intersection:** This type class description describes a class that satisfies boolean combination (intersection) of other class descriptions.

4. **The union:** This type class description describes a class that satisfies boolean combination (union) of other class descriptions.

5. **The complement:** This type class description describes a class that satisfies boolean combination (complement) of another class description. It means the indicated class extension consists of those individuals that are NOT members of the class extension of the complement class.

6. **The subclass:** This describes a class as subclass of another class.

7. **The equivalent:** This asserts that two class descriptions involved have the same class extension (i.e., both class extensions contain exactly the same set of individuals).

8. **The disjoint:** It asserts that the class extensions of the two class descriptions involved have no individuals in common.

9. **The property restriction:** We explain this type of class description in constraint definition mechanisms.

**Relation definition mechanisms:** OWL uses the word “property” as a substitute for a relation. In OWL, a property description is the basic building blocks of property axioms.
(informally called property definitions). A property axiom defines the characteristics of a property (Dean et al., 2003; Smith et al., 2003; Patel-Schneider et al., 2004).

1. **The independent property:** In its simplest form, a property axiom just defines the existence of a property.

2. **The subproperty:** This defines that the property is a sub-property of some other property.

3. **The domain:** It asserts that the subjects (domain) of a property must belong to some other classes. For example, the domain of the property hasBankAccount can be either a Person or a Corporation.

4. **The range:** For a property one can define multiple range axioms. A range axiom asserts that the values of a property must belong to the indicated class or to data values in the specified data range. For example the property madeFromGrape has a domain of Wine and a range of WineGrape. That is, it relates instances of the class Wine to instances of the class WineGrape.

5. **The equivalent property:** It can be used to state that two properties have the same property extension.

6. **The inverse property:** Properties have a direction, from domain to range. In practice, people often find it useful to define relations in both directions: persons own cars, cars are owned by persons. This definition mechanism can be used to define such an inverse relation between properties.

7. **The functional property:** A functional property is a property that can have only one (unique) value y for each instance x, i.e. there cannot be two distinct values y1 and y2 such
that the pairs \((x, y_1)\) and \((x, y_2)\) are both instances of this property. For this purpose, OWL defines the built-in class owl:FunctionalProperty. For example the husband property is functional, \textit{i.e.,} a woman can have at most one husband.

\textbf{8. The inverse functional property:} If a property is declared to be inverse-functional, then the object of a property statement uniquely determines the subject (some individual). More formally if a property, \(P\), is tagged as InverseFunctional then for all \(x, y\) and \(z\):

\[ P(y, x) \text{ and } P(z, x) \text{ implies } y = z \]

For example for each object of biologicalMotherOf statements (some human) one should be able to uniquely identify a subject (some woman). Inverse-functional properties resemble the notion of a key in databases.

\textbf{9. The transitive property:} When one defines a property \(P\) to be a transitive property, this means that if a pair \((x, y)\) is an instance of \(P\), and the pair \((y, z)\) is also instance of \(P\), then we can infer the pair \((x, z)\) is also an instance of \(P\). If a property, \(P\), is specified as transitive then for any \(x, y\), and \(z\):

\[ P(x, y) \text{ and } P(y, z) \text{ implies } P(x, z) \]

For example since the Santa Cruz Mountains Region is located in the California Region, then it must also be located in the US Region, since located in is transitive.

\textbf{10. The symmetric property:} A symmetric property is a property for which holds that if the pair \((x, y)\) is an instance of \(P\), then the pair \((y, x)\) is also an instance of \(P\). If a property, \(P\), is tagged as symmetric then for any \(x\) and \(y\):

\[ P(x, y) \text{ iff } P(y, x) \]
A popular example of a symmetric property is the friend-Of relation. The domain and range of a symmetric property are the same.

**Instance definition mechanisms:** Individuals are defined with individual axioms (also called "facts"). There are four mechanisms for description of individuals in OWL as following (Dean et al., 2003; Smith et al., 2003; Patel-Schneider et al., 2004):

1. the property values of individuals
2. the same identity with another individual
3. the different identity with another individual
4. the different identity between all individuals

**Constraint definition mechanisms:** In OWL, constraints (restrictions) are applied on properties. A property restriction is a special kind of class description. It describes an anonymous class, namely a class of all individuals that satisfies the restriction. OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints.

A value constraint puts constraints on the range of the property when applied to this particular class description. A cardinality constraint puts constraints on the number of values a property can take, in the context of this particular class description. For example, we might want to say that for a soccer team the hasPlayer property has 11 values. For a basketball team the same property would have only 5 values.

**3.7 Summary**

This chapter introduces the interpretation of terms used in logical schema definitions as a source of semantic heterogeneity amongst databases. It defines the problem of
interpretation of the identifier symbols in the schema definitions as the focus of this work. The semantics of the identifier symbols depends on the interpretation of the people who formulate them. By assigning relevant terms, people can try to show their intention. Users of a language (i.e., database designers or programmers) are free to select their own terms (or symbols) for naming classes, attributes and so on, with no obligation to formally define their intention. The explicit formalization of the interpretation can help to deal with the semantic heterogeneity problem. Subsequently, we suggest ontologies as a solution for the semantic heterogeneity problem.

This chapter provides a definition of ontologies and shows why they are considered as a solution for the problem. Knowledge represented in ontologies is based on the conceptualization of communities. The implicit assumptions in a conceptualization are explicitly defined in the ontology in a way that reasoning systems can use them. There is a difference between conceptual schemata and ontology, conceptual schemata are based on application requirement specifications and are independent from implementation, while ontologies are built based on conceptualization with the minimum of application requirements being considered (Hakimpour, 2003).

Ontology conveys more knowledge than schema definitions in databases. Schemata are more concerned with the structural aspects of data representation and researchers have been working on the syntactic aspects of schema integration. While schemata are used for organizing data in databases, ontologies are concerned with the understanding of the members of communities. It is important to note that schema definitions are closely related to the definitions in ontology. Schema definitions contain knowledge about the ontology of a community. These definitions may also help to estimate and extract the ontology from the schemata. However, schema definitions should commit to domain ontology (Hakimpour, 2003).
It is important to state that ontology is different from thesaurus or a dictionary. A thesaurus or dictionary applies a limited number of known relations among terms (such as synonym and antonym) in contrast; an ontology describes the relationships among the concepts referred by terms. Ontologies define concepts and their attributes and also present existing relationship among concepts such as taxonomical relationship between concepts (Hakimpour, 2003).

Applying ontologies guarantees consistency in the communities’ understanding of statements made during a communication. However, they will not necessarily represent the complete conceptualization of every community committed to the ontology. By using ontologies, communities are able to communicate based on their defined ontologies and the complication of building ontologies is the cost of resolving ambiguity and semantics heterogeneity in communication (Hakimpour, 2003).

Ontologies are usually expressed in a logic-based language in order to obtain detail, accurate, consistent and meaningful description of the concepts, properties, and relations in a domain. A number of research groups in both America and Europe had already identified the need for a more powerful ontology modeling language. This leads to a joint initiative to define a richer language, called DAML+OIL. DAML+OIL in turn was taken as the starting point for the W3C Web Ontology Working Group in defining OWL, the language that is aimed to be the standard and broadly accepted ontology language of the Semantic Web. Semantic Web needs an ontology language that can formally describe the meaning of the terminologies used in Web documents. OWL has been designed to meet this need for a Web Ontology Language (Antonion and Harmelen, 2003).
Chapter 4: ONTOLOGY-BASED WEB DATA INTEGRATION SYSTEM
Chapter 4  Ontology-based Web Data Integration System

4.1. Introduction

We propose an approach and system architecture for the integration of semi-structured data sources, from the internet in this chapter. This approach uses ontologies for resolving semantic conflicts between web data sources. Resolving of semantic conflicts between user query terms and web sources terms are the main issue described in general in this chapter and in full detail in Chapter 5. The proposed approach for web data integration system architecture in this chapter was motivated from the OBSERVER (Mena et al., 2000). Our proposed approach has two main differences as compared to the OBSERVER approach in order to cover the main drawbacks and shortcomings of the OBSERVER and other reviewed web data integration approaches:

- **We use domain specific ontologies to create user query.** The domain specific ontology is developed by domain experts and consists of all existing terms and concepts in the specific domain. The user is obliged to use the terms from the domain specific ontology for his/her query, therefore the user needs to be familiar with the domain ontology. We propose one specific representation model for domain ontology and suitable GUI for user’s interaction with the domain ontology in order to pose queries in Chapter 5.

- **We use different approach and algorithm for mapping between user query terms and local ontology terms.** The major problem in ontology-based data integration systems is detecting similarities among ontologies and performing semantic mapping between ontologies. This is also a major problem for the OBSERVER. The OBSERVER tries to resolve this problem by an Inter-Ontology Relationships Manager module. OBSERVER merges ontologies based on existing similarity relations kept in the Inter-ontology
Relationships Manager. Afterwards, it performs semantic mapping between two ontologies. The merging process of web ontologies in OBSERVER is highly costly with respect to time because the system needs to frequently merge two web ontologies in mapping time (because web ontologies change frequently). In our research we propose a different approach for semantic mapping between web ontologies. In our approach, we map related part of two ontologies with user query terms based on query path (discussed in detail in Chapter 5). This reduces the run time. We also present a new ontology representation model which enhances the mapping quality. The proposed semantic mapping approach is discussed in full detail in Chapter 5.

We describe our proposed approaches for web data integration in Section 4.2. We discuss system architecture of web data integration in Section 4.3. We conclude this chapter in Section 4.4.

4.2 Web Data Integration Approach

In the integration of web data, the main issues appear in the following areas:

- Target web source selection
- Resolving of heterogeneity conflicts
- Translation of user query to web source query language
- Web-data extraction
- Data transformation and loading to company’s local data

In this section, we present our ontology-based solution for web data integration. The proposed web data integration system is based on a virtual approach for data integration,
on the fly integration for semantic schema integration and multiple ontology for ontology architecture.

4.2.1 System approach

The Ontology-based approach for resolving of semantic schema conflicts in the integration of web data sources is shown in Figure 4.1. This approach uses ontologies for resolving of semantic conflicts between web data sources. Our proposed approach uses domain specific ontologies for the creation of user queries. There is a domain specific ontology for each application area that covers the semantic definition of terms which are required for user queries in a particular application domain. The domain ontologies are modeled in a uniform representation model. The user can browse the domain ontology and choose terms for his/her query; afterwards the system creates the user query. We assume each web source has an underlying pre-existing local ontology on the web and each local ontology is associated with one or more web sources (shown in Figure 4.2).

After the creation of the user query, the web ontology server chooses local ontologies related and relevant to the user query domain and sends them to the mapping module. First, the local ontology is transformed to the system uniform representation model and then the user query terms are mapped to corresponding terms from the local ontology. Subsequently, the user query is rewritten using the corresponding terms of the local ontology. This system uses ontologies to resolve semantic schema conflicts between web data sources. The proposed system resolves semantic schema heterogeneities between web source and user query through semantic mapping between the domain ontology and the local ontology.
Fig. 4.1 System approach for resolving semantic schema conflicts
The rewritten user query is sent to a query process module for reformulation and creation of optimized query plan from the user query. The sub queries obtained from the reformulation process are translated to the web sources query languages by translators and then the answer to the sub-queries are extracted from related web sources by wrappers. Each wrapper knows the structure of the underlying web source and extracts the related data to its sub-query and presents the extracted data in a common format. Through wrappers, the system resolves data model heterogeneity conflicts between web data sources.

The extracted data obtained through wrappers are sent to the converter. The converter resolves data value heterogeneity conflicts between data values. The converter exploits conversion functions and rules for resolving heterogeneities between two data value (such as units, data types, value format conversion functions). Finally the query plan operations are executed and final answers are exported to the user’s preference format by the exporter.

Resolving of semantic conflicts between terms from user queries and the terms from web sources is a main issue in this system. The system resolves this problem by using a semantic map between the domain and local ontologies. A reasoning system first finds the similarities in the terms (those terms that exist in the user query) between the domain and local ontologies and subsequently maps the corresponding concepts to each other. For example, “People” in ontology A (shown in Figure 4.3) is mapped to “staff” in ontology B.
(as shown in Figure 4.4) or “Faculty” is mapped to “Academic staff”. In this way, we resolve semantic conflicts between ontologies.

How the corresponding terms and similarity relations between ontologies are found is a difficult and complex task and requires the execution of an algorithm (for detection of similarities), scripting language (for representing mapping information) and tools (for execution of semantic mapping). In Chapter 5 we propose our semantic mapping algorithm for detecting similar terms between the domain and local ontologies and mapping them to each other.

Our system implements a query based approach to information extraction and integration, from heterogeneous and distributed web data sources. The extraction and integration process in the proposed system consists of eleven major tasks as follows:

1. Creation of user query;
2. Determination of related local ontologies and their underlying web sources with query domain;
3. Transformation of related local ontologies to internal uniform representation model;
4. Semantic mapping between query terms and related local ontologies terms;
5. Rewriting of user query with corresponding terms from local ontologies;
6. Reformulation of query and creation of optimized query plan;
7. Translation of sub queries to web wrapper query languages;
8. Extraction of data;
9. Conversion of data values;
10. Execution of query plan operations;
11. Exporting of data;

Our proposed web data integration system covers all abstraction levels of data heterogeneity conflicts between web data sources. The system applies:

- domain and local ontologies as a solution for resolving schema heterogeneities (as shown in Figure 4.5);
- wrapper as solution for resolving data model heterogeneities;
- converter as solution for resolving data value heterogeneities;

The proposed web data integration system is scalable to any domain by adding a related domain ontology to the system. That means in order to use the system in any application area we must develop and add a domain ontology (relevant to application domain area) to system.

Fig. 4.5 Proposed ontology architecture for resolving semantic schema conflicts
4.3 System Architecture

The elements of the system architecture are illustrated in Figure 4.6 and described in detail in the remainder of this section.
4.3.1 Query Construction module

This module consists of two tasks as discussed below. These tasks involve users directly. A GUI facilitates this user involvement.

1) Selection of Domain Ontology. At this stage, a domain ontology corresponding to a user’s query domain is selected. This domain specific ontology contains all the terms needed to express the semantics of the user query. The user then examines the ontology and selects the terms that correspond to the meaning of the terms of his/her query. It is important that the domain ontology is precise and complete for the relevant domain.

For example, consider the following user query:

“The names of faculty staff at computer department of the University of Malaya”.

The System receives the query and chooses the university ontology as the domain specific ontology. We assume ontology B, as shown in Figure 4.4 is the partially university ontology in the system and is the chosen domain specific ontology.

Next, the user browses ontology B and selects “Academic staff” and “Computer science department” concepts from this ontology that correspond with the concepts of “faculty staff” and “computer department” in his/her query.

2) Creation of query. After selecting the domain ontology and selecting terms from the domain ontology, the query construction module builds a query based on the system query language.

4.3.2 Web ontology server

This module consists of the following tasks (Mena et al., 2000):

1) Information about local ontology. This module provides information about local ontologies residing on the internet. The web ontology server maintains a catalog of ontologies, which contains information about local ontologies and the data sources underlying each local ontology.
The web ontology server can be invoked by other system modules in order to obtain information about the structure of any ontology on the internet. The web ontology server can be used for the following services:

- Obtaining existing terms from the ontology
- Obtaining definitions of terms
- Graphical representation of the ontology

2) Selection of local ontology. After the Query Construction module has created the user query, the web ontology server is invoked in order to select the local ontologies that are related to the domain of the user’s query. The main task of the web ontology server is to select the local ontologies that correspond to the domain of the user’s query.

4.3.3 Semantic mapping module

The aim of this module is to map terms from the user query (chosen from the domain specific ontology) to the terms in the local ontology. The terms in the user query is replaced with the terms from the local ontology. For this purpose, first the chosen local ontology is transformed to the system uniform representation model. We propose one specific uniform representation model for representation of ontologies (please refer to Chapter 5). Subsequently the semantic mapping module finds semantically corresponding terms of user query from the local ontology and replaces the user query terms with the local ontology terms. This replacement thus, resolves any semantic conflicts between the user query terms and the local ontology terms and also helps to resolve semantic heterogeneity conflicts between user query terms and web data sources. This module is the focus of the next chapter.

There are three main challenges in semantic mapping as stated below:

- How to find semantic mappings between two ontologies;
• How to represent and describe the semantic mapping information;
• How to execute the mappings;

Generally, in the ontology semantic mapping process, first, similarity relations between terms of two ontologies are detected and extracted. Afterwards, similarity relations between terms are represented and formalized by one description language. These similarity relations (called semantic mapping information) are used for semantic mapping between the terms of two ontologies. We use following general example in order to clarification of semantic mapping concept:

The general tasks for semantic mapping between ontology A and B (shown in Figures 4.3 & 4.4) can be as follows:

1. Extraction of similar concepts between two ontologies: “computer department” with “computer science department”, “people” with “staff”, “Faculty” with “Academic staff”, “Staff” with “Technical staff”, and so on are similar to each other.

2. Measurement and determination of type of similarity relations between terms: Each approach and algorithm may consider different types of similarity relations between terms (such as “Equivalent”, “Less general”, “More general” or “Overlapping”; for example, “computer department” has equivalent relation with “computer science department”, “Master Program” has less-general relation with “Graduate Program” and “Lecturer” has overlapping relation with “Associate Professor”).

3. Representation of similarity relation between terms: In this step, similarities between terms are formalized. For example, we should represent the similarity relation between “Faculty” with “Academic staff” by one formal language. These
formal descriptions from similarity relations are called semantic mapping information.

4. Execution of semantic mapping between similar concepts: In this stage, the concepts are similar to each other, are mapped together. For example, “computer department” is mapped to “computer science department”.

Ontology semantic mapping is a difficult and complex process and require the execution of an algorithm (for detection and measurement of similarities), scripting language (for representing mapping information) and tools (for execution of semantic mapping). In Chapter 5 we discuss our approach for ontology semantic mapping and propose an ontology representation model and an algorithm for mapping of user query terms to local ontology terms.

4.3.4 Query process module

This module consists of the following tasks:

1) Reformulation of user query. After the mapping process, the user query that has been rewritten into terms of one or more local ontologies is reformulated into one or more sub queries. The query process module must first reformulate the query into sub queries that refers directly to the schemas in the web sources. In order for the system to do this, it needs to have a set of source descriptions. The description of data source specifies its contents, attributes, constraints on its contents. This information exists in the web ontology server.

For example user has query about specifications of one product. If some of product specifications exist in website A and some in website B, so query process module reformulates user query to two sub-queries A and B.

2) Creation and optimization of the query plan. After the minimal set of data sources were selected for a given query and the query was reformulated to some sub queries a key problem is to find the optimal query execution plan for these sub queries. In particular, the
query plan specifies the order in which to perform the different operations between the sub queries (join, selection and projection). Typically, the optimizer selects a query execution plan by searching a space of possible plans and comparing their estimated cost.

4.3.5 Extraction module

This module consists of the following tasks:

1) **Translation of user query to wrapper query language.** After the query was reformulated into one or more sub queries and the query plan was created by the query process module (discussed in Section 4.3.4), each sub query is translated to the local web wrapper query language.

2) **Data extraction:** data is extracted from the pre-selected web sources by wrappers. Wrappers are used for retrieving data from web sources. A wrapper is a module which understands a specific data organization (Chawathe et al., 1994; Fileto, 2001). It knows how to retrieve data from the underlying repository and hide the specific data organization to the rest of the information system.

4.3.6 Data correlation module

This module consists of the following tasks:

1) **Conversion of data values.** The extracted data obtained through wrappers are sent to the converter. The converter resolves data value heterogeneity conflicts between extracted data values. The converter exploits conversion functions and rules for resolving heterogeneities between two data value (such as units, data types, value format conversion functions).

2) **Correlation of answers.** After wrappers have extracted an answer of each sub query and answers were converted, the query plan operations are next executed and query-answers are correlated.
3) **Representation of answer.** This module represents extracted data based on one common data model for users. The extracted data from different sources is represented based on common data format for users.

### 4.4 Summary

This chapter describes an approach and system architecture for the integration of semi-structured data sources, from the internet. This approach uses ontologies for resolving semantic conflicts between web data sources. Resolving of semantic conflicts between user query terms and web sources terms are the main issue described in this chapter. The Semantic mapping module resolves this problem through the use of ontologies.

The proposed approach uses domain specific ontologies for the creation of user queries. There is a domain specific ontology for each application domain that covers all the semantic definition of existing terms in a particular application domain. Users can browse the domain ontology and choose terms and concepts for his/her query, the system then creates a user query based on the system query language. We assume each web source has one or more underlying pre-existing local ontology and each local ontology is associated with one or more web sources.

User query terms are mapped to the local ontology terms and the user query is subsequently rewritten using the terms from the local ontology. Finally, the user query is translated to the web source query language and the data is extracted, converted and presented to the user.

Our proposed web data integration system covers all abstraction levels of data heterogeneity conflicts between web data sources. The system applies:
• ontology as a solution for resolving schema heterogeneities;
• wrapper as solution for resolving data model heterogeneities;
• converter as solution for resolving data value heterogeneities;

The proposed web data integration system is scalable to any domain by adding a related domain ontology to the system. That means in order to use the system in any application area we must develop and add a domain ontology (relevant to the application domain area) to the system.
Chapter 5:
SEMANTIC ONTOLOGY MAPPING
Chapter 5 Semantic Ontology Mapping

5.1. Introduction

The major aim of our work is to give a solution for resolving semantic schema heterogeneities in a web data integration system. For this purpose we first recommended a system architecture for web data integration in Chapter 4. The proposed system uses ontologies for resolving semantic conflicts between web data sources. The proposed system resolves semantic heterogeneities between web sources and user query through semantic mapping between the user query terms (chosen from domain ontology) and local ontologies. This semantic mapping also helps to translate queries into the web sources query languages and subsequently, resolves semantic schema conflicts between user query terms and web data source terms.

In the web context, many web data sources are involved and the number of involved web data sources changes frequently; web designers and users are free to use their own terms and vocabulary and schemata which are subject to frequent changes. These features of the web make it difficult to create a global, integrated or merged ontology in order to aid semantic integration. Therefore, our system uses inter-mappings between domain and local ontologies for semantic integration between user query terms and web data sources terms.

The extraction and integration process in the proposed system consists of eleven major tasks as follows:

1. Creation of user query;
2. Determination of related local ontologies and their underlying web sources with domain of query;
3. Transformation of related local ontologies to internal uniform representation model;
4. Semantic mapping between query terms and related local ontologies terms;
5. Rewriting of user query with corresponding terms from local ontologies;
6. Reformulation of query and creation of optimized query plan;
7. Translation of sub queries to web wrapper query languages;
8. Extraction of data;
9. Conversion of data values;
10. Execution of query plan operations;
11. Exporting of data;

In this chapter, based on our research objectives we focus on semantic mapping and query construction modules of the proposed system. We suggest an approach for the creation of user query and an approach for resolving semantic schema conflicts between user query terms (chosen from domain ontology) and related local ontologies terms. Our approaches for creation of user query and mapping algorithm cover the first, fourth and fifth tasks of the integration process mentioned above. We clarify our interpretation of domain and local ontology in Section 5.2. We propose a uniform representation model for modeling of ontologies in Section 5.3. We discuss the approach for query construction in Sections 5.4 and 5.5. We clarify some of the major ontology mapping concepts and ontology heterogeneities in Section 5.6 and its subsections. We present our solution for semantic ontology mapping in Section 5.7 and evaluate proposed mapping algorithm in Section 5.8. Finally, Section 5.9 discusses the summary of this chapter.

5.2 Domain and Local Ontologies

Our proposed system uses ontologies as a solution for reconciliation of semantic schema heterogeneities between web data sources. The system utilizes two types of ontologies:
domain specific ontology and local ontology. A domain ontology is created for one specific domain. For example, if the system has been developed for the domain of a university, so one university specific ontology is designed and created for the system. The users of the system choose their query-terms from the domain ontology and are not free to use other preferred terms. Terms used in user queries are restricted to the terms in the domain ontology.

There are many ontologies on the web that are used for the semantic description of web data. We call these ontologies, local ontologies. In our system we assume any web source has an underlying local ontology. Therefore, in order to resolve semantic conflicts between user query terms (chosen from the domain ontology) and the terms from web resources, a semantic mapping between the user query terms and the related local ontologies terms is needed. This semantic mapping relates similar terms from the two different sources by specifying the correspondence between them.

5.3 Ontology Uniform Representation Model

The mapping module of the system finds similar and corresponding terms between the related local ontology and domain ontology and maps them to each other. Local ontologies on the web have been formalized in different models and languages. In order to compare and find similar terms between the domain and local ontology, the system needs to represent all ontologies in a uniform model. In our system we propose one uniform representation model for ontologies. This representation model is general and any ontology with any representation model can be transformed to this uniform representation model.

**Definition 1:** $T=(C,A,R,V)$, each ontology element (term) is one of following entities:
• C: concept or instance of one concept
• A: attribute of one concept
• R: relationship between concepts
• V: constraint over a concept

For example student (concept), age (attribute), master student (it is considered as sub-concept of student in our model), attend (relationship between student and class) and “<20” (value range of “max-credit-course” relationship as constraint over student) are some element (term) of university ontology.

Definition 2: \( C=(\text{name}, \text{syn-set}, A, \text{key-property}) \), each concept is defined with its name, set of its synonyms, attributes and its key properties. The key properties are subset of concept attributes and relationships. The key properties are specific specifications of one concept that characterize the concept. These key properties are specified just for concept definitions of the domain ontology during the development of the domain ontology. We use these properties as a mapping criterion for finding similar terms in our mapping algorithm.

Definition 3: \( A=(\text{name}, \text{syn-set}) \), attribute is defined with a name and a set of synonyms.

Definition 4: \( R=(\text{name}, \text{syn-set}, \text{domain}, \text{range}) \), each relationship is defined with a name, set of synonyms, domain and range.

Definition 5: \( V=(\text{value or } C) \), this feature is used for representing the constraint over a concept that is either a value or a concept. It begins with one of these characters: “=”, “<”, “>”, “<=”, “=>” or “<>” and one value (string) or a concept (C).

Definition 6: \( O=(G, G') \), each ontology is represented by two graphs.

Definition 7: \( G=(N, E), N=<C>, E=<is-a> \), \( G \) is acyclic directed rooted graph that consists of nodes and edges. Each node is a concept (or instance of a concept). Each edge is “is-a” relation that shows sub-concept (subclass) relation between nodes; \( G \) is a
hierarchy concept model of ontology. Each node has one father and may have none, one or more child node. If one node has two fathers, the model resolves this problem with repeating child node for each one of its fathers.

Definition 8: $G' = (N, E')$, $N = \langle C, V \rangle$, $E' = \langle R \rangle$, $G'$ is cyclic graph that consists of nodes and edges. Each node is a concept (or instance of a concept) or one value. Each edge is a relationship between two nodes that shows the relationship between concepts. $G'$ is a concept relationship model of ontology.

In a uniform representation model, all elements (concepts, attributes, relationships and values) are string (chain of characters). Our representation model and formalization of ontology is general, so our proposed approach which uses this formalization will work with any ontology representation languages. We need to transform (in the mapping module of the system) the local ontology to the uniform representation model. This representation model represents the main exploitable information in an ontology.

We use the table structure (Relational Database) to store ontologies represented in the uniform representation model using any DBMS implementation such as MySQL (MySQL, 2003). We illustrate the required tables for storage of ontologies in the uniform representation model form as follows:

Table 5.1 Concept Hierarchy table

<table>
<thead>
<tr>
<th>CONCEPT-HIERARCHY Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept-Id</td>
</tr>
</tbody>
</table>

Table 5.1 is used for storage of concept hierarchy model of ontology (G graph). Concept-id, Father-id and all Att-Ids in this table are foreign key. Concept-id and Father-id refers to CONCEPT table and all Att-Ids refer to ATTRIBUTE table.
Table 5.2 Concept table

<table>
<thead>
<tr>
<th>Concept-Id</th>
<th>Concept-Name</th>
<th>Syn-Name&lt;sub&gt;1&lt;/sub&gt;</th>
<th>….</th>
<th>Syn-Name&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
</table>

Table 5.2 is used for storage of all concept elements and their synonyms. Concept-Id is primary key of this table.

Table 5.3 Attribute table

<table>
<thead>
<tr>
<th>Att-Id</th>
<th>Att-Name</th>
<th>Syn-Name&lt;sub&gt;1&lt;/sub&gt;</th>
<th>….</th>
<th>Syn-Name&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
</table>

Table 5.3 is used for storage of existing attributes and their synonym sets in an ontology. Att-Id is primary key of this table.

Table 5.4 Relation table

<table>
<thead>
<tr>
<th>Rel-Id</th>
<th>Rel-Name</th>
<th>Syn-Name&lt;sub&gt;1&lt;/sub&gt;</th>
<th>….</th>
<th>Syn-Name&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
</table>

Table 5.4 is used for storage of existing relationships and their synonym sets in an ontology. Rel-Id is primary key of this table.

Table 5.5 Concept Relationship table

<table>
<thead>
<tr>
<th>Relation-Id</th>
<th>Domain-Id</th>
<th>Range-Id</th>
<th>Value</th>
</tr>
</thead>
</table>

Table 5.5 is used for storage of concept relationship model of ontology (G’ graph). Relation-Id in this table is a foreign key to RELATION table; Domain-Id and Range-Id are foreign keys that refer to the CONCEPT_HIERARCHY table.
Table 5.6 Key Properties table

<table>
<thead>
<tr>
<th>Concept-Id</th>
<th>KeyAtt-Id_1</th>
<th>......</th>
<th>KeyAtt-Id_n</th>
<th>KeyRel-Id_1</th>
<th>......</th>
<th>KeyRel-Id_n</th>
</tr>
</thead>
</table>

Table 5.6 is used for storage of key properties of concepts. Concept-Id in this table is foreign key to CONCEPT table; KeyAtt-Ids are foreign keys that refer to the ATTRIBUTE table and KeyRel-Ids are foreign keys that refer to the RELATION table.

5.4 User Query Construction

Our proposed approach is a domain specific approach because the user is confined to choosing terms of a specific domain ontology for his/her query. This approach can be extended for any domain so that the relevant domain ontology would be previously developed. The user is confined to use just one domain ontology for his/her query. Users cannot pose complex queries because the query construct and structure is based on the system query structure and based on elements of the internal (system) uniform representation model of the domain ontology. We define the following structure and syntax as system query structure and syntax for the expression of the user query.

```
SELECT <attributes names>
FROM <concept name>
WHERE { <attribute names: values> FROM <concept name_1>
          <attribute names: values> FROM <concept name_2> .... }
```

In this query structure, the user can query the attributes of only one concept from the domain ontology (we call it concept query). That means each user query possesses only one concept query. The user can specify constraints and conditions on his/her query.
Constraints and conditions are expressed after the “WHERE” clause in the query expression. We clarify the query syntax and structure with the following example:

Suppose that a user needs the names and emails of professors in Law at universities who are above 50 years of age and are female. This query is in the university domain, so we assume there is a university specific ontology in the system. So the user traverses terms in the university ontology and chooses his/her query terms. Afterwards, the system constructs an expression based on the user query as follows:

```sql
SELECT name, Email 
FROM Professor 
WHERE name = Law FROM Department 
   sex = Female FROM Professor 
   age > 50 FROM Professor 
```

Note that the user cannot pose complex queries. The user can split his/her complex query to simple sub-queries and subsequently submit them to the system. For example, if a user has a query about attributes relating to two concepts, then he/she must pose two separate queries to the system. The user can simply specify conditions for the attributes of concepts which are in the query path. The path through which a user traverses in the domain ontology graphs for reaching his/her query terms is called a query path. For example in the above query, conditions which have been specified for the attributes of name, sex and age all are related to the concepts in the query path. We discuss query paths further in the next section.

---

### 5.5 GUI and Query Path

In a query construction process the user traverses the internal uniform representation graphs of the domain ontology in order to choose his/her query terms. The path through
which a user traverses in the domain ontology graphs for reaching his/her query terms is called the query path. We use this query path as comparison and mapping criteria for finding similar and corresponding terms between the user query and the local ontologies in the semantic mapping algorithm.

A user interacts with the system through the GUI (Graphical User Interface) in the query construction module of the system. The GUI must display the domain ontology to the user so that the user can find his/her query terms easily and quickly. In the time the user takes to interact with the system for the construction of the query, the system performs two tasks: first creation of query and second specifying of query path for use in the semantic mapping algorithm.

How we design the GUI and display the graph models of domain ontology to the user, have a role in the precise and exact creation of the query path. In our approach, the GUI first displays a list of domain ontologies to the user. The user chooses one domain ontology related to his/her query. Then, the GUI displays super concepts of domain ontology to the user (major concepts in top levels of concept hierarchy graph). The user chooses one of the super concepts. This chosen concept is the root of the query path. We call it $C_1$. In next step the GUI shows three types of information to the user related to $C_1$ that are choices for the user for a second term of query path (we call $C_2$):

- Sub-concepts (children) of $C_1$,
- Attributes of $C_1$,
- Relationships and their ranges which $C_1$ is the domain of those relationships.

Figure 5.1 provides a partial illustration of this information about the concept of student.
In this step:

- If a query has a condition or a constraint on the attributes of $C_1$, then the user enters the constraint value of the attributes in the value fields and

- If this concept is a query concept in which a user needs to query its attributes then: choose “Y” for query attributes, afterwards

- In order to choose next concept of query path (we call it $C_2$), The user follows one of these steps :
  - Choose one sub-concept; or
  - Choose range concept (not range value) of one relationship;

The above mentioned step is repeated in order to choose other query path terms. In this way the query construction module obtains the query path and finally creates the user query. The Query path contains all the terms that the user traverses in the above mentioned steps. The query path does not consist of any values for the attributes but just the attribute names. There is just one query concept in the query path; which mean the user can just query attributes of a concept (as shown in Figure 5.2). For example the query path for query mentioned in Section 5.4 from the university ontology can be:

<table>
<thead>
<tr>
<th>Sub concepts of Student:</th>
<th>Attributes of student:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diploma student</td>
<td>Value</td>
</tr>
<tr>
<td>Bachelor student</td>
<td>- webpage</td>
</tr>
<tr>
<td>Master student</td>
<td>- program</td>
</tr>
<tr>
<td>PhD student</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Query</td>
</tr>
<tr>
<td></td>
<td>Y/N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>study</td>
<td>department</td>
</tr>
</tbody>
</table>

Fig. 5.1 Example of GUI of university ontology
We consider the following definition for a query path:

**Definition 9:** $\mathcal{QP} = (\mathcal{N}_p, \mathcal{E}_p, \mathcal{E}'_p)$, a acyclic directed rooted path from uniform representation graphs $(G & G')$ of the domain ontology traversed by a user.

**Definition 10:** $\mathcal{N}_p = (\text{C-name, } A_1\text{-name, } A_2\text{-name}...)$, is a concept or instance name (C) and its attached attributes specified by user as a constraint or query attribute in query construction time.

**Definition 10:** $\mathcal{E}_p = (\text{is-a})$, shows sub-concept relationship between concepts in query path.

**Definition 11:** $\mathcal{E}'_p = (\text{R-name})$, represents the relationship name.

\[
\begin{align*}
\mathcal{C}_1(A_1, A_2,...) & \xrightarrow{E_{1p}} \mathcal{C}_2(A_1, A_2,...) & \xrightarrow{E_{2p}} & \ldots \xrightarrow{E'_{i-1p}} \mathcal{C}_i(A_1, A_2,...) & \xrightarrow{E'_{ip}} & \ldots \\
\mathcal{E}_{i-1p} & \xrightarrow{E_{ip}} & \mathcal{C}_{i}(A_1, A_2,...) & \xrightarrow{E_{jp}} & \ldots & \mathcal{C}_{j+1}(A_1, A_2,...)
\end{align*}
\]

Fig. 5.2 Query path

### 5.6 Ontology Mapping

After creating the user query and query path through the query construction module, the query needs to be translated to the relevant web source wrapper query languages. For this translation, first, the user query terms must be semantically mapped to similar terms in the local ontologies which are related to the relevant web sources. This semantic mapping relates each query term with its semantically similar terms in the related local ontology. The semantic mapping is partially performed between the domain ontology (related to query path terms) and the local ontology. Therefore our semantic mapping algorithm is in
the context of ontology mapping. We need an algorithm in order to semantically map the domain and local ontologies.

In the next sections we focus on related issues in ontology mapping and then propose a semantic mapping algorithm. We start with the definition of ontology mapping and then introduce the problem of existing heterogeneities between ontologies, which is characterized by different kinds of mismatches between ontologies. We need to resolve these mismatches in our mapping process in order to obtain a high quality result.

5.6.1 Definition of ontology mapping

In the ontology-related research literature, the concept of *mapping* has a range of meanings, including integration, unification, merging, mapping, etc. We consider below definitions that are compatible with our usage of this term.

- In (Madhavan et al., 2002), *it is defined that a mapping will be a set of formulae that provide the semantic relationships between the concepts in the models.*

- In (Noy and Musen, 2000), *it is said that Mapping is to establish correspondences among the source ontologies, and to determine the set of overlapping concepts, concepts that are similar in meaning but have different names or structure, and concepts that are unique to each of the sources.*

- In (Rahm and Bernstein, 2001), *it is defined: Given two ontologies O1 and O2, mapping one ontology onto another means that for each entity (concept C, relation R, or instance I) in ontology O1, we try to find a corresponding entity, which has the same intended meaning, in ontology O2.*

Consequently, we define ontology mapping, in the context of our research, as a process of determining a set of correspondences and similarities that identify similar elements (terms) in different ontologies. A well defined mapping process can be considered as a component which provides a mapping service. This service can be used in various applications. For
example, an ontology integration application can use the finding mappings as the first step towards an integrated ontology. Two tasks have to be conducted in the ontology mapping process. One is to find the correspondences and similarities between ontology elements and the other is to describe and define the discovered mappings so that other components could make use of them (Xiaomeng, 2004).

The following example (shown in Figure 5.3) illustrates an example mapping. Two ontologies $O_1$ and $O_2$ describe the partial domain of computer department. A reasonable mapping between the two ontologies is given in Table 5.7.

![Diagram of example ontologies $O_1$ and $O_2$](image)

**Fig. 5.3 Example ontologies $O_1$ and $O_2$**

**Table 5.7 Mapping table for ontologies $O_1$ and $O_2$**

<table>
<thead>
<tr>
<th><strong>Ontology $O_1$</strong></th>
<th><strong>Ontology $O_2$</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer department</td>
<td>Computer science department</td>
</tr>
<tr>
<td>Undergraduate Program</td>
<td>Under Graduate Program</td>
</tr>
<tr>
<td>Master Program</td>
<td>Graduate Program</td>
</tr>
<tr>
<td>People</td>
<td>Staff</td>
</tr>
<tr>
<td>Faculty</td>
<td>Academic Staff</td>
</tr>
<tr>
<td>Staff</td>
<td>Technical Staff</td>
</tr>
<tr>
<td>Lecturer</td>
<td>Assistant Professor</td>
</tr>
<tr>
<td>Professor</td>
<td>Professor</td>
</tr>
</tbody>
</table>
5.6.2 Ontology mismatches

The heterogeneities between ontologies are called ontology mismatches and are divided into: language level mismatches and ontology level mismatches (Klein, 2001).

2) **Language Level Mismatches:** Mismatches at the language level occur when ontologies written in different ontology languages are combined. In (Klein, 2001), four types of mismatches are identified as follows:

- **Syntax:** Different ontology languages often use different syntaxes. For example, to define the concept of car in RDF Schema, one uses `<rdfs:Class ID = "Car">`. In LOOM, the expression `(defconcept Car)` is used to define the same class.

- **Logical representation:** This mismatch is about the difference in representation of logic notions. For example, in some languages, it is possible to state explicitly that two classes are disjoint (e.g. disjoint A B), whereas it is necessary to use negation in subclass statements in other languages (e.g. A subclass-of (NOT B), B subclass-of (Not A))

- **Semantics of primitives:** This mismatch is about the difference in the semantics of language constructs. Despite the fact that sometimes the same name is used for a language construct in two languages, the semantics may differ, e.g., there are several interpretation of A equalTo B.

- **Language expressivity:** This difference implies that some languages are able to express things that are not expressible in other languages. For example, some languages have constructs to negation, whereas others have not.

3) **Ontology Level Mismatches:** Mismatches at the ontology level happen when two or more ontologies that describe partly overlapping domains are combined. The ontology level mismatches occur because of differences in conceptualization and explication of ontologies. A conceptualization or explication mismatch is a
difference in the way a domain is interpreted or specified (Visser, 1997). The major
ontology level mismatches are as follows (Klein, 2001):

- **Domain of description**: This is a mismatch in the level of detail to which that
domain is modeled. For example in the university domain one ontology might
represent staff but only classify them into a few categories, while another ontology
might make further distinctions between categories of staff.

- **Terminology**: two ontologies may use different names to represent the same term
(e.g. "price" and "cost"), or the same name to represent different terms.

- **Paradigm**: Different paradigms can be used to represent terms (e.g. "date of birth"
and "age").

- **Description model**: Several choices can be made for the modeling of terms in the
ontologies. What is modeled as an attribute in one ontology may be modeled as an
concept (class) in another ontology. For example "Author" as attribute of "book" in
one ontology and "author" as a concept that has relationship with "book" in a
second ontology and "Author" as a relationship in a third ontology).

Figure 5.4 shows the framework of ontology heterogeneities and mismatches. As
recommended by (Visser, 1997), mapping of terms at the ontology level is a task that
requires understanding of the meaning of terms, and cannot be fully automated. In order to
enhance the quality and accuracy of ontology mapping we need to consider all mismatches
in the mapping approach and resolve them. We can evaluate one mapping approach based
on how many of the above mentioned ontology mismatches are covered and solved by the
approach. In our ontology mapping approach, we consider the mismatches mentioned
above and try to resolve them in an algorithm. However resolving all mismatches is so
difficult and some are infeasible for automated support and need human intervention to be
solved. In order to integrate ontologies, it is important to distinguish mismatches that are hard to solve, and those that are not (Klein, 2001).

5.7 Query-based Ontology Mapping Approach and Architecture

Our purpose of ontology mapping is to find similar terms between user query terms (chosen from domain ontology) and relevant local ontology and map these user query terms to the local ontology terms. This ontology mapping helps to resolve semantic conflict between the user query and web data source. The semantic mapping module finds semantically similar terms and replaces the user query terms with their corresponding terms from the local ontology. This replacement thus, resolves any semantic conflicts between the user query terms and the local ontology terms. Hence, the semantic mapping process helps to resolve semantic schema heterogeneities between the user query terms and terms from the web data sources.

Fig. 5.4 Framework of ontology heterogeneities
There are three main challenges in semantic mapping as stated below:

- How to find similar terms between the domain ontology and the local ontologies (called mapping pair elements);
- How to represent and describe the similar relationships (called mapping information);
- How to execute the mappings;

In this section we focus on these challenges and present an approach and architecture for semantic mapping module. We consider both the quality and run-time complexity of the mapping results.

5.7.1 Terminology

Before we discuss the ontology mapping approach that we are proposing, we need to clarify the terminology we will use. It is almost impossible to find an agreement on the exact meaning of terms. We use definitions as stated in (Xiaomeng, 2004) which are consistent with the definitions and descriptions we found elsewhere.

- **Integrating.** Creating a new ontology from two or more existing ontologies (ontologies are in different subject).
- **Merging.** Creating a new ontology from two or more existing ontologies (ontologies are in same subject).
- **Mapping.** Relating similar terms from different sources to each other (according to some metric) by specifying the correspondence between them.
- **Transforming.** Changing the representation formalism of an ontology while preserving the semantics.
- **Ontology element.** Terms exist in ontology and are referred by information sources elements. In general ontology element can be a concept, attribute, relation or instance that has been represented in ontology.
• **Mapping pair element.** Each mapping pair element specifies certain element of ontology $O_1$ that is mapped to certain element in ontology $O_2$ (specify of semantically correspond elements).

• **Similarity measure.** The degree of how much two elements are similar to each other. Similarity measure is usually one number within the range of $[0 \ 1]$.

• **Similarity threshold.** A number within the range of $[0 \ 1]$. If the similarity measure between two elements is above the threshold then two elements are called similar.

• **Mapping expression.** Each mapping expression express how elements in one mapping pair element are related to each other (express of semantic of similarity between two elements).

• **Mapping algorithm or function.** An algorithm or function takes two ontologies $O_1$ and $O_2$ as input and returns mapping pair elements and expressions between those two ontologies as output.

• **Mapping criteria.** The mapping algorithm compares two elements for finding mapping pair elements based on some criteria such as name, data type and description of elements.

• **Mapping local cardinality.** Shows mapping cardinality into one mapping pair element. It indicates how many terms from two sources appear in one mapping pair element. It consists of 1:1, 1:n, n:1 or n:m cardinalities.

### 5.7.2 Mapping module architecture

The components of the mapping module are illustrated in Figure 5.5 and described in detail in the remainder of this section.
5.7.2.1 Transformer

As mentioned previously there are two kinds of mismatches between ontologies that should be resolved in an ontology mapping process, namely, language level and ontology level mismatches. Local ontologies are represented with different ontology representation models and languages. Each ontology language is different from other ontology languages in the syntax and semantic of language constructs and formalism mechanisms. These differences make it difficult to compare and map ontologies to each other. The solution is to transform different ontology representations to a uniform representation model. That means the system uses one uniform internal representation model for all ontologies. For this purpose the transformer components transform the local ontology languages to a system uniform representation model and then store them in the repository. Indeed the transformer components resolve language level mismatches between the domain and local ontologies. The domain ontologies are modeled in the uniform representation model and stored in the repository in advance.

5.7.2.2 Mapper

The mapper maps semantically the user query path terms to the corresponding terms in the local ontologies. This semantic mapping relates each query path terms with its
semantically similar and corresponding term of related local ontology. The mapping is performed between a part of the domain ontology (related to query path terms) and the local ontology.

We can evaluate an ontology mapping approach based on two main factors: the quality of the mapping results and the run time complexity of the mapping algorithm. The quality of the mapping indicates the accuracy of similarity between discovered mapping pair elements. Run time complexity indicates the run time and performance of the mapping algorithm. Runtime complexity shows how long the algorithm discovers mapping pairs elements and maps them to each other. Some approaches have a high quality mapping result but they are not applicable because the mapping algorithm has a high run time. Therefore we must consider both factors when designing mapping algorithms.

There are various approaches for semantic mapping between ontologies that have been proposed by other researchers. Some of the recent researches and approaches in the ontology mapping domain as thoroughly discussed in Chapter 2 are MAFRA (Silva and Rocha, 2003), Chimaera (McGuinness, et al., 2000), PROMPT (Noy and Musen, 2003; Noy and Musen, 2001; Noy and Musen, 2000), QOM (Ehrig and Staab, 2004), GLUE (Doan, et al., 2003) and ASCO (Thanh-Le et al., 2004). Our approach was motivated by some of the ideas of the above researches. The approaches mentioned exploit available information from ontologies and map similar terms of two given ontologies to each other by using mapping algorithms. We will compare our proposed mapping approach with the above approaches in Section 5.8.

5.7.2.3 Semantic mapping algorithm

Inputs of our mapping algorithm are: query path, domain ontology and local ontology. There are three types of ontology element (term) in a query path: concept (C), attribute (A)
and relationship \( R \). All elements (terms) are strings (chain of characters) and may be a word, term or expression (combination of words). The first node of the query path is the root. The query path consists of the user query terms and some non-query terms. The un-query terms are terms which have been traversed by user for reaching to query terms. These terms do not exist in the user query.

The purpose of our mapping algorithm is to find local ontology terms that are semantically similar with the terms in the query path and then rewrite the user query with the corresponding similar terms. In order to calculate the similarity between two terms from a query path and the local ontology, the following function is used in our mapping algorithm:

**MF (Mapping Function):** \( MF(T_1, T_2) = [0 \ 1] \); This function calculates the similarity between two terms. A value range of \([0 \ 1]\) indicates the degree of similarity. The MF performs two sub-functions for similarity calculation. The first sub-function normalizes a term to simple tokens. In this sub-function each term (concept, attribute or relationship) is:

- **Tokenized:** this task eliminates characters such as ‘-‘, ‘_’, ‘,’ ‘.’, ‘:’ from a composed term and converts term to separate and simple tokens. For example ‘Admission-Fee’ is converted to ‘Admission’ & ‘Fee’.
- **Lemmatized:** this task extracts singular term from a term. For example ‘Fees’ is converted to ‘Fee’.
- **Eliminated:** this task deletes words like ‘of’, ‘for’, ‘by’ from a composed term and converts term to separate and simple tokens. For example ‘SubjectofProgram’ is converted to ‘Subject’ & ‘Program’.

For normalization, our algorithm exploits and uses one domain specific dictionary. This dictionary consists of all existing terms in a specific domain including their synonym sets.
The second sub-function compares gained tokens (string without space) from normalization of terms with each other and calculates the similarity between tokens. Finally, the similarity between two terms is calculated from the aggregation of their token similarities. There are well-known metrics for calculating string similarities between two tokens such as Jaro-Winkler metric (Winkler, 1999), Levenstein metric and Monger-Elkan (Cohen et al., 2003). We apply the Jaro-Winkler metric in implementation of our mapping algorithm.

The main steps of our mapping algorithm are as follows (consider the structure of the query path in Figure 5.2):

**First step:** Name Matching; Root of query path $C_1(name)$ and all its synonyms names $C_1(syn-name)$ are compared with all the local ontology concepts $C_L(name)$ (shown in Figure 5.6).

![Diagram of Name Matching](image)

Fig. 5.6 Name Matching

```
for all $C_L(name)$ <> null do
    {If $MF[C_1(name), C_{iL}(name)] >= threshold$ then
        similarity-table $\leftarrow (C_1, C_{iL})$;
        Else: for all $C_1(syn-name)$ <> null do
            {If $MF[C_1(syn-name), C_L(name)] >= threshold$ then
                Add $(C_1, C_{iL})$ to similarity-table; exit; }
    }
```

The result of the above step is some similar terms pairs, which have a similarity measure above the algorithm threshold. We call them, candidate mapping pairs. We store the candidate mapping pairs in a table (we call similar table). In the next steps (second, third
and fourth steps) the algorithm must find the best similar pair among candidate mapping pairs. If algorithm does not find the similar concept of root \(C_1\) then it stores pair \(<C_1,\text{null}>\) in the C-mapping-table. In this case the algorithm considers \(C_2\) (next concept of query path) as root and repeat first step with \(C_2\).

**Second step: Father Matching:** in this step father and grandfather of \(C_i\) are compared with father and grandfather of each \(C_{iL}\) \((C_{iL}\ \text{in similar table})\). For each \(C_{iL}\) we calculate:

\[
\begin{align*}
\text{If} \ (\text{father of } C_1 \text{ and } C_{iL} <> \text{null}) \text{ then:} & \\
F\text{ather-matching}(C_1,C_{iL}) & \leftarrow MF[\text{father}(C_1),\text{father}(C_{iL})]; \\
\text{If} \ (\text{grandfather of } C_1 \text{ and } C_{iL} <> \text{null}) \text{ then:} & \\
\{F\text{ather-matching}(C_1,C_{iL}) + = [MF(\text{grandfather}(C_1),\text{grandfather}(C_{iL})]; & \\
\text{Father-matching}(C_1,C_{iL}) /= 2; \} \\
\end{align*}
\]

**Third step: Key-Property Matching:** The algorithm executes the MF between key properties of \(C_1\) with all the attributes and relationships of \(C_{iL}\) \((C_{iL}\ \text{in similar table})\). The algorithm saves the number of matches between \(C_1\) and each \(C_{iL}\).

\[
\begin{align*}
\text{while } C_{iL} \ \text{name} <> \text{null} \ \text{do} & \\
\{ \text{for all } A,R \text{ of } C_{iL} \text{ do} & \\
\text{If each } MF[C_1(\text{key-property-name} & \text{ key-property-syn-names}), C_{iL}(A & R)] \geq \text{threshold} \text{ then} & \\
\text{C}_{iL}(\text{similar-property}) + 1; \} \\
\end{align*}
\]

**Fourth step: Aggregation:** In this step the algorithm finds the most similar corresponding concept with \(C_1\) among all \(C_{iL}\) from the similar table (shown in Figure 5.7). For this purpose, the algorithm aggregates the results of the father matching (second step) and key-property matching (third step) for each \(C_{iL}\). The \(C_{iL}\) which has the highest weight is the final mapping concept of \(C_1\). We call it \(C_{iL}\) and store it in the concept mapping table.

\[
\begin{align*}
\text{For all } C_{iL} \text{ do} & \\
\{ \text{Weight}(C_{iL}) \leftarrow \text{father-matching}(C_{iL}) + C_{iL}(\text{similar-property}); \} & \\
C_{iL} & \leftarrow C_{iL}[ \ \text{MAX}(C_{iL}-\text{weight}) > \beta] \\\n\text{C-mapping-table} & \leftarrow (C_1,C_{iL}); \\
\end{align*}
\]
**Fifth step:** After finding the similar concept of \( C_1 \), if \( C_1 \) has attribute in the query path then we must find its similar attributes in the local ontology. Notice that we just execute a MF between \( C_1\text{-attribute-name} \), all \( C_1\text{-attribute-synset-names} \) with attributes-names and relationships-names of its mapping pair (\( C_{1L} \) in mapping table). We choose a maximum MF that is above the threshold and store similar-attribute pairs in the \textit{att-mapping table} (such as: \(<C_1\text{-A}_1,C_{1L}\text{-A}_{1L}>, <C_1\text{-A}_2,C_{1L}\text{-A}_{2L}>, \ldots\)).

```plaintext
while \( C_1 \text{(A-name)}<>\text{null} \) do
  ( If \( MF(C_1 \text{(A-name)}, C_{1L} \text{(A-name or R-name)}) \geq \text{threshold} \) then
    Add \((C_1 \text{(A-name)}, C_{1L} \text{(A-name or R-name)})\) to \textit{att-mapping-table};
  Else:
    while \( A\text{-syn-name}<>\text{null} \) do
      ( If \( MF(C_1 \text{(A-syn-name)}, C_{1L} \text{(A-name or R-name)}) \geq \text{threshold} \) then
        Add \((C_1 \text{(A-name)}, C_{1L} \text{(A-name or R-name)})\) to \textit{att-mapping-table};
      Else \( i \leftarrow i+1; \) )
```

**Sixth step:** we must find the mapping concept for the next node (concept) of the query path \((C_2)\) (shown in Figure 5.8). There are two situations here: \( C_2 \) has “is-a” relationship with \( C_1 \) (\( C_2 \) is sub-concept of \( C_1 \)) or \( C_2 \) has “R” relationship with \( C_1 \) (\( C_1 \) and \( C_2 \) are domain and ranges of the same R).

**In the first situation**, the algorithm aggregates the results of the name matching (first step) and key-property matching (third step) to find the mapping concept of \( C_2 \). The difference is
the algorithm does not compare $C_2$ with all of the local ontology concepts as $C_2$ is a sub-concept of $C_1$. thus,

- Algorithm first compares (MF) $C_2$ with children of $C_{1L}$.
- If algorithm could not find similar concept of $C_2$ in children of $C_{1L}$ then it compares (MF) $C_2$ with siblings of $C_{1L}$.
- If algorithm could not find similar concept of $C_2$ in siblings of $C_{1L}$ then it compares (MF) $C_2$ with all concepts that are in the range of relationships in which $C_{1L}$ is the domain of those relationships.
- Finally if the algorithm could not find similar concept of $C_2$ then it compares (MF) $C_2$ with $C_{1L}$. Because $C_{1L}$ may be further general and cover the semantics of both of $C_1$ and $C_2$.

If the algorithm finds a similar concept of $C_2$ then it stores similar pair $<C_2, C_{2L}>$ in the $C$-mapping-table. If it does not find the similar concept of $C_2$ then it stores pair $<C_2, null>$ in the $C$-mapping-table. In this case the algorithm uses $C_{1L}$ instead of $C_{2L}$ in the next steps, because $C_{2L}$ is null.

**In the second situation** ($C_2$ has “$R$” relationship with $C_1$), the algorithm performs the following tasks:

The algorithm executes name matching (first step) between $R$, $R$-synonyms with all relationships of $C_{1L}$ ($C_{1L}$ is domain of relationships).

- If the algorithm finds a similar relationship then it:
  - Execute first, second, third and fourth steps of algorithm between $C_2$ and ranges of discovered similar relationship. If find similar concept of $C_2$ then enter similar pair $<C_2, C_{2L}>$ in $C$-mapping-table.
  - Else enter $<C_2, null>$ in $C$-mapping-table.
If the algorithm can not find the similar relationship with $R$ then the algorithm executes first, second, third and fourth steps of algorithm between $C_2$ and all of the local ontology concepts. If it finds a similar concept of $C_2$ then it enters the similar pair $<C_2, C_{2L}>$ in the C-mapping table else it enters $<C_2, null>$ in the C-mapping-table.

Fig. 5.8 Sixth step: finding next concept matching

After finding the similar concept of $C_2$, if $C_2$ has attribute in the query path then we must find its similar attributes in the local ontology. Notice that we just execute a MF between $C_2$-attribute-name, all $C_2$-attribute-synset-names with attributes-names and relationships-names of its mapping pair ($C_{2L}$ in mapping table). We choose a maximum MF that is above
the threshold and store similar-attribute pairs in the **att-mapping table** (such as: \(<C_2-A_1,C_{2L}-A_{1L}>\), \(<C_2-A_2,C_{2L}-A_{2L}>\)…..).

**Seventh step:** The algorithm is repeated as in the Sixth step for the next nodes (concepts) of the query path. If the algorithm encounters a query concept \((C_q)\) of a query path (query concept possesses query attributes), the following tasks are performed by the algorithm:

- If the algorithm finds similar concept of query concept \((C_q)\) from local ontology (based on previous step: step 6) then it must find similar attributes of the query concept in the local ontology. Notice that we just execute MF between \(C_q\)-\text{attribute-name}, all \(C_q\)-\text{attribute-synset-names} with \text{attributes-names} and \text{relationships-names} of its mapping pair. We choose the maximum MF that is above the threshold and store similar-attribute pairs in **att-mapping table** (such as: \(<C_q-A_1,C_{jL}-A_{1L}>\), \(<C_q-A_2,C_{jL}-A_{2L}>\)…..).

```java
while \(C_q\text{(A-name)}<>\text{null}\) do
{
    if \(\text{MF}[C_q\text{(A-name)},C_{jL}\text{(A-name or R-name)}] >= \text{threshold}\) then
        Add \((C_q\text{(A-name)},C_{jL}\text{(A-name or R-name)})\) to **att-mapping-table**;
    Else: while \text{A-syn-name} <>\text{null} do
        if \(\text{MF}[C_q\text{(A-syn-name)},C_{jL}\text{(A-name or R-name)}] >= \text{threshold}\) then
            Add \((C_q\text{(A-name)},C_{jL}\text{(A-name or R-name)})\) to **att-mapping-table**;
        } } }
```

- **Exceptional task 1:** If the algorithm cannot find similar attribute for any query attribute of \(C_q\) in the above step (called unfound similar attribute) then the algorithm executes Matching Function between unfound similar attribute of \(C_q\) and all concepts in the local ontology. That means the algorithm compares unfound similar attributes with all concepts from local ontology because an attribute of the domain ontology may be a concept term in the local ontology.

- **Exceptional task 2:** if the algorithm cannot find the similar concept of the query concept \((C_q)\) in the local ontology then the algorithm executes Matching Function between the query concept \((C_q)\) and all attributes and relationship terms in the local
ontology. That means the algorithm compares $C_q$ with all attributes and relationships from the local ontology because $C_q$ may be an attribute or relationship term in the local ontology. In this situation the algorithm waves from finding similar attributes of $C_q$. The algorithm returns null for attribute mapping pairs of attributes of the query concept ($C_q$). In this situation, if there is next node in the query path (after query concept), the algorithm searches all local ontology terms for finding the mapping concepts of next node of the query path.

- If the algorithm cannot find the similar term of the query concept ($C_q$) in the above steps the running of the algorithm will be stopped and mapping is not executed between the user query terms and the local ontology terms and this local ontology fails.

**Eighth step:** The algorithm rewrites the user query with the discovered mapping concepts and attributes. If all user query terms are mapped to the local ontology terms then we have a complete query mapping, otherwise, we have a partial query mapping. In the case of partial query mapping, the query process module of the system decides to continue or terminate the process of partially mapped user query.

We clarify our mapping algorithm with following examples:

**Example 1:** We assume the query construction module creates the following user query and query path according to the user’s question:

```
SELECT name, Email
FROM Professor
WHERE {
  name = Law FROM Department
  sex = Female FROM Professor
  age > 50 FROM Professor }

School → Department (name) → Staff → Academic-Staff → Professor (sex, age, name?, Email?)
```
In the above user query, “Professor” is a query concept, “Email” and “name” are query attributes, “Department” is the first query constraint concept with “name” constraint attribute and “professor” is a second query constraint concept with “age” and “sex” constraint attributes.

We assume the user query and query path are created on the following partial domain ontology (Figure 5.9):

![Sample Partial domain ontology A](image)

We assume the web ontology server chooses the following local ontology (Figure 5.10):

![Sample partial local ontology B](image)

As an example, we consider the following features for the query path terms:

- syn(school): college, faculty
  - key-property(school): null
- \text{syn(department)}: \text{faculty}, \text{school}
  \text{key-property(department)}: \text{null}
  \text{Constraint attribute}: \text{name}

- \text{syn(staff)}: \text{people}
  \text{key-property(staff)}: \text{salary}, \text{work}
  \text{syn(salary)}: \text{null} \& \text{syn(work)}: \text{employ}

- \text{syn(academic staff)}: \text{faculty}
  \text{key-property(academic staff)}: \text{null}

- \text{syn(professor)}: \text{Prof.}
  \text{key-property(professor)}: \text{teach}
  \text{syn(teach)}: \text{null}
  \text{Constraint attribute}: \text{name, age, sex, email}

  \text{syn(age)}: \text{date of birthday}

Outputs of the algorithm are C-mapping table and attribute-mapping (shown in Table 5.8 & 5.9).

\begin{table}[h]
\centering
\caption{Concept mapping table}
\begin{tabular}{|l|l|}
\hline
\textbf{Query concept} & \textbf{Local Ontology concept} \\
\hline
School & College \\
Department & Department \\
Staff & People \\
Academic staff & Faculty \\
Professor & Professor \\
Lecturer & Lecturer \\
\hline
\end{tabular}
\end{table}
Example 2: We assume the query construction module creates the following user query and query path according to the user’s question:

```sql
SELECT web-page, field
FROM master-program
WHERE

    { name = computing FROM Faculty }

Program → Postgraduate-program → Master-program (web-page?, field?)
Offer from School (name)
```

In the above user query, “master-program” is a query concept, “web-page” and “field” are query attributes, “school” is the query constraint concept with “name” constraint attribute.

We assume the user query and query path are created on the following partial domain ontology (Figure 5.11):

<table>
<thead>
<tr>
<th>Query attribute</th>
<th>Ontology attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Sex</td>
<td>Sex</td>
</tr>
<tr>
<td>Age</td>
<td>Date of birthday</td>
</tr>
<tr>
<td>Email</td>
<td>Email</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
</tbody>
</table>

Table 5.9 Attribute mapping table

![Fig. 5.11 Sample Partial domain ontology C](image-url)
We assume the web ontology server chooses the following local ontology (Figure 5.12):

As an example, we consider the following features for the query path terms:

- syn(university): university, college, college
- syn(Program): degree-program, course
- syn(postgraduate-program): program
- syn(master-program): postgraduate-program, master

attribute: web-page, field

syn(web-page): URL, homepage

syn(field): name, title

- syn(school): faculty, department

attribute: web-page, name

syn(web-page): URL, homepage

syn(name): title

Outputs of the algorithm are C-mapping table and attribute-mapping (shown in Table 5.10 & 5.11).
5.8. Evaluation of Ontology Mapping Approach

We presented a comparison framework in Chapter 2 (Section 2.7) that characterizes and classifies ontology mapping approaches. We characterize the specifications of our mapping algorithm based on this comparison framework as follows (shown in Table 5.12):

<table>
<thead>
<tr>
<th>Mapping function specification</th>
<th>Our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of mapping function</td>
<td>Multiple</td>
</tr>
<tr>
<td>Number of mapping criterion</td>
<td>Multiple</td>
</tr>
<tr>
<td>Combination of mapping function</td>
<td>Composition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping algorithm technique</th>
<th>Our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping technique</td>
<td>Linguistic-based</td>
</tr>
<tr>
<td></td>
<td>Constraint-based</td>
</tr>
<tr>
<td></td>
<td>Path-based</td>
</tr>
<tr>
<td>Mapping granularity</td>
<td>Element-level</td>
</tr>
<tr>
<td>Mapping element</td>
<td>Concept, instance, attribute</td>
</tr>
<tr>
<td>Mapping cardinality</td>
<td>1:1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input &amp; output</th>
<th>Our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology representation model</td>
<td>Graph stored in relational data-based</td>
</tr>
<tr>
<td>Auxiliary information</td>
<td>domain specific dictionary</td>
</tr>
<tr>
<td>Input of algorithm</td>
<td>User query path, domain ontology, local ontology</td>
</tr>
<tr>
<td>Output of algorithm</td>
<td>Concept mapping table, attribute mapping table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User interaction</th>
<th>Our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Selection of query and query path terms, approve or disapprove of answer</td>
</tr>
</tbody>
</table>

Table 5.10 Concept mapping table 2

<table>
<thead>
<tr>
<th>Query concept</th>
<th>Local Ontology concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Null</td>
</tr>
<tr>
<td>Postgraduate-program</td>
<td>Postgraduate-program</td>
</tr>
<tr>
<td>Master-program</td>
<td>Master</td>
</tr>
<tr>
<td>School</td>
<td>Department</td>
</tr>
</tbody>
</table>

Table 5.11 Attribute mapping table 2

<table>
<thead>
<tr>
<th>Query attribute</th>
<th>Ontology attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web-page</td>
<td>URL</td>
</tr>
<tr>
<td>Field</td>
<td>Null</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
</tbody>
</table>
- Number of mapping function: multiple; our mapping algorithm uses multiple mapping functions for finding similar terms (concept, attribute and relationship mapping function).

- Number of mapping criterion: multiple; each mapping function use multiple mapping criteria for comparison of similarity of two elements (It uses $C(name)$, $C(syn-set)$, $C(key-Property)$ as mapping criteria for concepts, $R(name)$ and $R(syn-set)$ as mapping criteria for relations, $A(name)$ and $A(syn-set)$ as mapping criteria for attributes.

- Combination of mapping functions: composite; our mapping algorithm employs multiple mapping functions with multiple mapping criteria (composite).

- Mapping granularity: It is in element level not structural level. Because it finds similarity in element granularity (concept, attribute and relationship similarity).

- Mapping technique: It is linguistic-based because it finds similarities by element-name matching and element-synonym matching (string matching). It is also constraint-based because it uses key properties (key-A and key-R) of concepts for finding best similarity.

- It is path-based because it uses path of query. It considers semantic relations in paths and restricts domain of search in local ontology.

- Mapping element: Concept, instance, attribute; mapping function maps concept, instance and attribute terms to each other.

- It has 1:1 mapping local cardinality. Because it maps each query term to one local ontology term no more.

- Ontology representation model: graph; our mapping approach uses graph as data structure model for representation of ontology and relational database for storage of
ontology representation model. It can be applied for any ontology model or language because it uses one general representation model for ontologies.

- Auxiliary information: It uses domain specific dictionary as auxiliary information for normalization of terms and help in finding similarities.
- Input of Mapping algorithm: query path, domain ontology and local ontology are inputs of our mapping algorithm.
- Output of algorithm are c-mapping-table (concept mappings) and att-mapping-table (attribute mappings).

We can evaluate an ontology mapping algorithm based on two main factors: quality of mapping results and run time complexity of the mapping algorithm (Klein, 2001; Mitra and Wiederhold, 2002). In rest of this section we discuss about quality and runtime of proposed mapping algorithm.

**5.8.1 Quality of algorithm results**

We can evaluate the quality of one mapping approach based on how many of the ontology mismatches are covered and solved by the approach. As mentioned in sub-section 5.6.2 the ontology mismatches are categorized in the language level and the ontology level. The ontology level mismatches consist of the domain of description, Terminology, Paradigm and description model mismatches. Our mapping approach resolves:

- Language level mismatches by transforming ontologies to the system internal uniform representation model.
- Domain of description mismatches by synonym set matching. Step 6 of algorithm also helps to resolve domain of description mismatches.
- Terminology mismatches by the synonym set matching, key property matching and father matching.
• Paradigm mismatches by synonym set matching. Key property matching and father matching also help to resolve paradigm mismatches.

• Description model mismatches by the exceptional tasks 1 and 2 (step 7 in the algorithm).

It is difficult to resolve all mismatches and some are not feasible for automated support and need human intervention to be resolved (Klein, 2001). The following features are strength points of proposed algorithm. We used the following techniques and features for enhancement of the mapping quality and help to resolve mismatches between ontologies in the proposed algorithm:

• **Posing user query based on domain ontology terms:** This resolves semantic conflicts in presentation layer of system (user level) because different users use same terms for their queries and they possess same view of domain and application area through domain ontology.

• **Modeling ontologies based on Uniform representation model:** The existing definition mechanisms in uniform representation model help to resolve mismatches. The synonym set and key property features in the proposed uniform representation model help to enhance the quality of the mapping result. As mentioned above these features help to resolve domain of description, terminology and paradigm mismatches between ontologies. The proposed representation model is general and possesses main exploitable information in an ontology.

• **Using query path in mapping algorithm:** The query path is the most effective mapping technique of the proposed algorithm. The algorithm searches the domain ontology in order to find the user query concept and its query attributes through the query path. The query path gives two advantage points to the algorithm as follows:
1. *Reduce runtime of each achievement of the mapping results:* It directs the algorithm towards the query concept and its attributes and reduces the search domain hence reducing the time for the algorithm to search.

2. *Gain higher quality mapping results:* The query path possesses concepts which have some semantically relation with query concept because all of concepts of a query path exist on a same path of ontology graph and terms on a same path are related to each other through is-a or R relationship. Therefore algorithm has further information about meaning of query concept through query path that help to find corresponding term with query concept. This causes higher quality mapping results.

We implement our mapping algorithm in .NET framework and present the experimental results related to the quality of algorithm in Chapter 6.

### 5.8.2 Runtime complexity of algorithm

In this section we calculate the complexity levels which occur when doing mapping of query path terms to terms of local ontology (Please refer to appendix-F for further details). We waived the complexity of the ontology transformation and assume the local ontology uses the system uniform representation model. We consider n as the number of elements (entities) in a local ontology (ontology size). The runtime of a single entity or fixed set and length of entities is independent of the size of the ontology and we consider complexity of O(1). Any step which requires access to the whole path of an ontology graph (acyclic rooted graph) leads to a complexity of O(log n) (depth of a path) and any step needs access to the whole ontology results in the complexity of O(n).

The proposed mapping algorithm maps the query path terms (chosen from the domain ontology by user) to the corresponding terms of the local ontology. We calculate the
Complexity of each step of the algorithm in the worst case as follows (shown in Table 5.13):

- **Complexity of MF (Mapping Function) used in algorithm:** We used Jaro-Winkler algorithm (Winkler, 1999) for Mapping Function (MF). The runtime complexity of MF depends on the string length of two terms and is $O(L_1 \cdot L_2)$ where $L_1$ and $L_2$ are length of two terms. In our algorithm we consider a maximum fixed length for terms. Therefore the complexity of MF is $O(MF) = O(1)$;

- **Complexity of First Step, Name Matching:** In this step the root of the query path is compared with all the local ontology terms. Therefore the complexity of this step is $O(n) \cdot O(MF) = O(n) \cdot O(1) = O(n)$;

- **Complexity of Second Step, Father Matching:** In this step the father and grandfather of candidate mapping terms are compared with the father and grandfather of the root of the query path. In the worst case we may have $n$ candidate mapping terms (all local ontology terms). Therefore the complexity for comparison of the father is $O(n) \cdot O(MF) = O(n)$ and for grandfather is $O(n) \cdot O(MF) = O(n)$. Total complexity for this step of algorithm is $O(n) + O(n) = O(n)$;

- **Complexity of Third Step, Key-Property Matching:** In this step key properties of concepts are compared with all attributes and relationships of candidate mapping terms. In the implementation of the algorithm we consider a fixed number of key property for each concept element of the domain ontology. The number of attributes and relationships for each concept element in the local ontology is independent of the ontology size. For the implementation of this step we consider a maximum fixed number of attributes and relationships for each concept element in order to reduce the runtime complexity of the algorithm. So the comparison of
attributes and relationships possesses the complexity of O(1). The number of candidate mapping terms may be n in the worst case. Thence the complexity in the worst case is: O(1).O(MF).O(n) = O(n);

- **Complexity of Fourth Step, Aggregation:** In this step we calculate the similarity weight. In the worst case the number of candidate mapping terms is “n” so the complexity is O(n);

- **Complexity of Fifth Step, Attribute Matching:** This step compares attributes of a concept with all attributes and relationships of the mapping term. In the worst case we can consider m attributes and relationships so the complexity of this step is O(m);

- **Complexity of Sixth Step, Next Concept Matching:** In the worst situation of this step we need to execute step 1, 2, 3 and 4 between next concept of the query path and all concepts of the local ontology. Therefore complexity of next concept mapping is the total complexity of steps 1, 2, 3 and 4 as follows:

\[ O(n) + O(n) + O(n) + O(n) = O(n); \]

The sixth step of algorithm is repeated in order to find mapping terms for all other next query path terms. The complexity of this iteration depends on the length of the query path. The domain ontology uses acyclic directed rooted graph which we can consider “log n” for query path size. Therefore the total complexity of the sixth step is O(n . log n);

- **Complexity of the Seventh Step, Query Concept Matching:** In the worst situation of this step we need to execute steps 1, 2, 3 and 4 between the query concept and all concepts of the local ontology. Therefore the complexity of this step is:

\[ O(n) + O(n) + O(n) + O(n) = O(n); \]
- **Complexity of step 8, query rewriting:** In this step, the algorithm rewrites the user query with discovered corresponding terms from the local ontology. This step has complexity of O(1).

Aggregation of algorithm complexity regards to mentioned above complexities is:

\[ O(n)+O(n)+O(n)+O(n)+O(m)+O(n \cdot \log n)+O(n)+O(1) = O(n \cdot \log n) \]

<table>
<thead>
<tr>
<th>Algorithm step</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>O(n)</td>
</tr>
<tr>
<td>Step 2</td>
<td>O(n)</td>
</tr>
<tr>
<td>Step 3</td>
<td>O(n)</td>
</tr>
<tr>
<td>Step 4</td>
<td>O(n)</td>
</tr>
<tr>
<td>Step 5</td>
<td>O(m)</td>
</tr>
<tr>
<td>Step 6</td>
<td>O(n \cdot \log n)</td>
</tr>
<tr>
<td>Step 7</td>
<td>O(n)</td>
</tr>
<tr>
<td>Step 8</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Table 5.13 Algorithm complexity

As stated in (Ehrig and Staab, 2004) QOM (Quick Ontology Mapping) (Ehrig and Staab, 2004) possesses complexity of O(n log n) and has the lowest and effective runtime complexity among mapping algorithms such as NOM (Ehrig and Sure, 2004), Prompt (Noy and Musen, 2000), Anchor-Prompt (Noy and Musen, 2001) and GLUE (Doan, et al., 2003). As calculated above our proposed algorithm has equal runtime complexity with the QOM algorithm.
5.9 Summary

In this chapter, we propose an approach and algorithm to resolve semantic schema conflicts in a web data integration system. First we introduce an uniform graph-based representation model for ontologies and then propose an approach for the creation of user queries based on this representation model. Subsequently, we propose an algorithm to semantically map terms from user queries to terms in the ontologies. Our mapping algorithm employs the user query path in order to find the similarities between user query terms and the local ontology terms.

Our mapping algorithm resolves:

- Language level mismatches by transforming ontologies to the system internal uniform representation model.
- Domain of description mismatches by synonym set matching. Step 6 of the algorithm also helps to resolve the domain of description mismatches.
- Terminology mismatches by the synonym set matching, key property matching and father matching.
- Paradigm mismatches by synonym set matching. Key property matching and father matching also help to resolve paradigm mismatches.
- Description model mismatches by the exceptional tasks 1 and 2 (step 7 in the algorithm).

The proposed algorithm exploits the following features and techniques to enhance the quality of the mapping results and reduce the runtime of the algorithm:

- **Posing user query based on domain ontology terms:** This resolves semantic conflicts in the presentation layer of the system (user level) because different users
use the same and corresponding terms for their queries and they possess same view of domain and application area.

- **Modeling ontologies based on Uniform representation model:** The existing definition mechanisms in uniform representation model help to resolve mismatches.

- **Using query path in mapping algorithm:** The algorithm searches domain ontology in order to find user query concept and its query attributes through query path. The query path gives two advantage points to algorithm as follows:

  1. *Reduce runtime of each achievement of mapping results:* It directs algorithm toward query concept and its attributes and causes to reduce the search domain of algorithm. As stated in Section 5.8 the runtime of algorithm possesses complexity of $O(n \log n)$.

  2. *Gain higher quality mapping results:* The query path possesses concepts which have some semantically relation with query concept. Therefore the algorithm has further information about meaning of query concept that helps to find corresponding term with query concept with higher quality mapping results.
Chapter 6: IMPLEMENTATION and ASSESSMENT of ONTOLOGY MAPPING APPROACH
Chapter 6 Implementation and Assessment of Ontology Mapping Approach

6.1 Introduction

In this chapter we focus on the implementation and experimentation of our semantic ontology mapping approach (proposed in Chapter 5). In Chapter 5, we proposed the ontology mapping module for performing semantic mapping between user query terms and local ontology terms. The proposed mapping module possesses the following components (please refer to Chapter 5: Figure 5.5):

- **Inputs** consist of query path, domain ontology and local ontology;
- **Mapper**: this component transforms local ontology to the system uniform representation model and stores the transformed local ontology in a relational database.
- **Mapper**: mapper consists of mapping functions which maps query terms to the local ontology terms;
- **Repository**: repository is used to store
  - Uniform representations of domain and local ontologies and
  - Mapping pair elements for reusing
- **Domain dictionary**: domain dictionary consists of information about existing terms in a specific domain and synonym set of terms.
- **Output**: output consists of two tables:
  - Concept mapping table which possesses concept mapping pair elements
  - and
Attribute mapping table which possesses attribute mapping pair elements.

In this chapter we focus on the implementation of our ontology mapping approach. For this purpose, we present a methodology in order to build the domain ontologies and create a sample domain specific ontology in the university domain and represent it with our uniform representation model in Section 6.2. We create a graphical user interface for the presentation of the domain ontology (university ontology) in Section 6.3. In Section 6.4, we present two sample local ontologies. We use these local ontologies for the evaluation of our mapping algorithm. In Section 6.5, we implement our proposed mapping algorithm and execute the implemented algorithm on the sample constructed domain and local ontologies (university domain ontology, University Science Malaysia and University Technology Malaysia local ontologies) and evaluate the experimental results of our mapping algorithm. Finally in Section 6.6, we discuss the conclusion of this chapter.

6.2 Building of University Domain Ontology

Our proposed web data integration system is query based and domain specific. That means the output of the system is an answer to the query which the user has posed in a specific domain area based on domain ontology terms. In order to extend this system to any other domain it is compulsory to create a domain ontology that consists of all required terms and semantics in that domain. This ontology is called the domain specific ontology. We need well-defined domain ontologies to successfully implement our mapping algorithm. The quality of mapping algorithm results has direct relation with completeness and accuracy of domain ontology. Therefore, we first discuss the methodology for building domain ontologies based on our uniform representation model.
6.2.1 Methodology for building domain ontology

There are approaches for creation of ontologies that have been proposed by researchers.
Some of these approaches in the ontology creation domain are (Uschold, and King, 1995; Uschold, 1996; Lopez, 1999; Noy and McGuinness, 2001). We refer reader to (Lopez, 1999) for discussion about ontology creation approaches and problems of them.

As mentioned in (Noy and McGuinness, 2001) there are some fundamental rules in ontology design that we should note:

- There is no one correct way to model a domain—there are always viable alternatives.
- Ontology development is necessarily an iterative process.
- Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest.

We will certainly need to revise the initial ontology. This process of iterative design will likely continue through the entire lifecycle of the ontology. We suggest the following phases for creation of ontologies based on our proposed ontology representation model (shown in Figure 6.1).

**Phase 1. Determination of domain and Scope of ontology**: We start the development of an ontology by clarifying its domain and scope. That is, answer several basic questions (Noy and McGuinness, 2001):

- What is the domain that the ontology will cover?
- For what we are going to use the ontology?
- For what types of questions will the information in the ontology provide answers?
- Who will use and maintain the ontology?
We should stay away from application domain requirements and try to be as general as possible. This is due to the fact that ontologies describe universal intension of terms for a community (Noy and McGuinness, 2001).

**Phase 2. Concept (or instance) extraction:** Based on the defined scope we extract existing and required concepts in conceptualization environment for the domain of interest.

**Phase 3. Concept hierarchy creation:** we arrange concepts and instances in a taxonomic (sub-concept, super-concept) hierarchy. If a concept A is a super-class of concept B, then every instance of B is also an instance of A. In other words, the concept B represents a concept that is a “kind of” A. There is no single correct concept hierarchy for any given domain. The hierarchy depends on the possible use of the ontology, the level of the detail that is necessary for the application, personal preferences, and sometimes requirements for compatibility with other models (Noy and McGuinness, 2001).

**Phase 4. Attribute and relationship extraction:** Based on the defined scope we extract relevant attributes of each extracted concept and existing relationships between extracted concepts in conceptualization environment for the domain of interest.

**Phase 5. Key-property determination:** we determine key properties for each extracted concept. Key properties of each concept are chosen from its attributes and relationships. Key properties define the role and main semantic of concepts. For example “teaching” is key property for “lecturer” concept, that shows essence and semantic of “lecturer”.

**Phase 6. Synonym determination:** we specify the synonyms for each concept, attribute and relationship. The synonym of each term is other name that is used by other communities for representing the same term.

**Phase 7. Modeling:** we modelize the extracted terms in a representation model. We can use an edit ontology tool such as Protégé-2000, Ontolingua, or Chimaera as ontology-editing environments for this purpose.
Phase 8. *Implementation:* Finally we implement and store representation model of ontology in a data base or file.

![Ontology creation steps diagram](image)

Fig. 6.1 Ontology creation steps

A practical question often asked is, “Whose role is it to build an ontology?” the person building an ontology should have a good understanding of the vocabulary and the conceptualization of the domain. Such knowledge helps to ensure the accordance of ontologies with the community’s conceptualization as a measure of quality for ontologies (Hakimpour, 2003).

6.2.2 Building of university ontology

We follow our suggested methodology for building the University domain ontology.

*Phase 1: Domain and Scope clarification.* The scope of our domain ontology is a university.

*Phase 2: Concept (or instance) extraction.* For extraction of existing concepts in the university, we need to investigate and study the conceptualization of this domain in detail. We use some information sources for this purpose such as knowledge of people who are in the university (such as staff, students), university websites, university specific vocabularies and other universities’ ontologies (such as the SHOE university ontology).

*Phase 3. Concept hierarchy creation:* We create a concept hierarchy from extracted concepts in a taxonomic form. We present the created concept hierarchy in a taxonomy form in Figure 6.2.
Fig. 6.2 Concept hierarchy diagram of an University domain ontology
**Phase 4. Attribute and relationship extraction:** We determine the attributes of each extracted concept and relationships between concepts.

**Phase 5. Key-property determination:** We use dictionaries to specify key properties of each concept. These key properties show the main meaning and semantic of each concept which characterizes a concept from others. We can discover them in definition expressions of each concept in the dictionary.

**Phase 6: synonym determination.** We use dictionaries and universities websites to specify synonyms of each extracted term (concepts, attributes and relationships).

In the following section we present results of the above mentioned phases. First, we present the created concept hierarchy in a taxonomy form. We present extraction attributes for each concept inside [brackets], synonyms inside {braces} and key properties inside (parentheses). Afterwards, we present relationships between concepts and the set of synonyms of attributes and relationships.

**Concept hierarchy:**

University [[University college] [home-page, name, country]
College {Faculty, School, Academic-Department} [home-page, name]
Department [Academic-Department] [home-page, name]
Laboratory {Lab} [home-page, name]
Staff {employee, worker, people, educational-employee} [home-page]
Academic-staff {Faculty, Academic-people, faculty-member, academician, academic} [home-page]
Professor [Prof] [URI] (key-relationship: teach)
Assistant-Professor [Assist-Prof] [name, Email, research-interest, home-page]
Associate-Professor [Assoc-Prof] [name, Email, research-interest, home-page]
Full-Professor [Professor, Prof] [name, Email, research-interest, home-page]
Visiting-Professor [Adjunct Professor] [name, Email, research-interest, home-page]
Lecturer (key-relationship: teach)
Full-time-Lecturer [Lecturer, Senior-lecturer] [name, Email, research-interest, home-page]
Part-time-Lecturer [name, Email, research-interest, home-page]
Visiting-Lecturer [Adjunct Lecturer] [name, Email, research-interest, home-page]
Gusting-Lecturer [Adjunct Lecturer] [name, Email, research-interest, home-page]
Post-Doctor [Postdoctoral, Post-doctorate] [name, Email, research-interest, home-page] (key-relationship: teach)
Tutor [instructor, trainer, educator] [name, Email, home-page] (key-relationship: teach)
Research-Assistant [RA, research-fellow][name, Email, research-field, home-page](key-relationship: research-field)
Teaching-Assistant [TA] [name, Email, home-page]
Researcher [research-worker, scientist] [name, Email, Research-field, home-page] (key-relationship: research-field)
Administrative-Staff [Non-Academic-Staff, management-staff]
University-Dean [Chair] [name, Email, home-page]
Deputy-Dean [dean- assistant]
Academic-Deputy-Dean [academic-dean-assistant, deputy-dean] [name, Email, home-page]
Development-Deputy-Dean [development-dean-assistant, deputy-dean] [name, Email, home-page]
Head-Department [name, Email, home-page, Department-name]
Head-Library [name, Email, home-page]
Administrative-Assistant [administrator]
Assistant-Registrar [name, Email]
Clerical-Assistant [name, Email]
Official-Assistant [name, Email]

Student [current-student, prospective student][home-page] (key-relationship: study-in)
Undergraduate-student
Diploma-Student [home-page]
Bachelor-Student [degree student][home-page]
Postgraduate-Student [Graduate-Student]
Master-Student [home-page]
Doctorate-Student [PhD-Student] [home-page]
Postdoctoral-Student [Post-doc-student] [home-page]
No-graduating-Student [home-page]

Research-group [Research] [home-page, Research-field] (key-relationship: Research-field)
Project [Project-Title, home-page] (key-relationship: Project-Title)
Center [unit] [Name, home-page]
Institute [academic centre][Name, home-page]
Program [Degree-Program, Academic-program, Educational-Degree] [home-page] (key-relationship: list-of-course)
Undergraduate-Program [web-page]
Diploma-Program [web-page, Field]
Bachelor-Program [web-page, Field]
Postgraduate-Program [Graduate-Program] [web-page]
Master-Program [Master-of-science] [web-page, Field]
Doctorate-Program [PhD, Doctor of philosophy] [web-page, Field]
Postdoctoral-Program [postdoctoral-position, postdoctoral-fellowship] [web-page, Research-field]
No-graduating-Program [web-page, Field]
Student-Exchanging-program [web-page, Field]

Admission [academic-admission][home-page]
Undergraduate-Admission [degree-admission][web-page, Field, Requirement, Admission-date] (key-attribute: requirement)
Postgraduate-Admission [web-page, Field, Requirement, Admission-date] (key-attribute: requirement)

Tuition-Fee [admission-fee, fee, charge, fee-structure] [home-page]
Diploma-Fee [diploma-charge] [web-page, amount] (key-attribute: amount)
Bachelor-Fee [bachelor-charge] [web-page, amount] (key-attribute: amount)
Master-Fee [master-charge] [web-page, amount] (key-attribute: amount)
Doctorate-Fee [PhD-charge] [web-page, amount] (key-attribute: amount)
Financial-Aid [financial-assistance][home-page]
Undergraduate-Financial-Aid [web-page]
Scholarship [web-page, amount] (key-attribute: amount)
Fellowship [web-page, amount] (key-attribute: amount)
Loan [web-page, amount] (key-attribute: amount)
Postgraduate-Financial-Aid [Graduate-Financial-Aid] [web-page]
Scholarship [web-page, amount] (key-attribute: amount)
Fellowship [web-page, amount] (key-attribute: amount)
Loan [web-page, amount] (key-attribute: amount)
Teaching-Assistantship [web-page, amount] (key-attribute: amount)
Research-Assistantship [web-page, amount] (key-attribute: amount)

About-university [Overview, about-us, about-college] [home-page]
Dean-Message [Chair-message, mission] [home-page]
History [home-page]
Map-Location [Place, get-there, how-get] [home-page]
Contact-Info [contact, phone-directory][home-page, address, Phone, fax, Email] (key-attribute: address, phone)
Publication [research-publication][home-page]
Article [web-page]
Journal-Article [Journal-publication, Journal-paper] [web-page, Title, publish-date, Author] (key-attribute: author)
Conference-Paper [conference-proceeding] [web-page, Title, publish-date, Author] (key-attribute: author)
Book [web-page, Title, publish-date, Author] (key-attribute: author)
Periodical [journal]
Journal [web-page, Title, publish-date, Author] (key-attribute: author)
Magazine [web-page, Title, publish-date, Author] (key-attribute: author)
Thesis [dissertation]
Doctoral-Thesis [PhD-thesis] [web-page, Title, publish-date, Author] (key-attribute: author)
Masters-Thesis [web-page, Title, publish-date, Author] (key-attribute: author)
Event \{news, event-news\} [home-page]

News [web-page, Description, Event-date]
Conference [web-page, Description, Event-date]
Seminar [web-page, Description, Event-date]
Workshop [web-page, Description, Event-date]

Career \{employment, recruitment, job-opportunity, vocation\} [home-page] (key-attribute: position)

Academic-Position \{Educational-position, lecturer-position\} [web-page, Position, requirement, deadline]
Postdoctoral-Position [web-page, Position, requirement, deadline] (key-attribute: Position)
Non-Academic-Position \{staff-position\} [web-page, Position, requirement, deadline] (key-attribute: Position)
Student-job-Position [web-page, Position, requirement, deadline] (key-attribute: Position)

Facility \{service\} [home-page]
Library [home-page]
Subject \{course\} [web-page, title]

Concept relationships:

Advisor (Academic-staff, Student)
Org-Publication (center, Publication)
Org-Publication (Institute, Publication)
Research-Publication (Research, Publication)
Project-Publication (Project, Publication)
Head (Center, Staff)
Head (Institute, Staff)
Head (Department, Staff)
Work (Staff, Department)
Teach (Academic-staff, Subject)
Member (Center, Staff)
Member (Institute, Staff)
Member (Department, Staff)
Member (Research, Staff)
Member (Project, Staff)
Offers (Department, Program)
List-of-course (program, Subject)
Study-in (student, program)

The synonym sets of attributes:

Webpage: \{URI, Web-address, URL, homepage\}
Name: \{Full-name, first-name, last-name, surname, given-name\}
Email: \{electronic-mail, mail\}
Phone: \{telephone, hand-phone, phone-number, Contact-number\}
Research-interest: \{research-field, research-area\}
Homepage: \{personal-webpage, personal-homepage, personal-page, URI, URL\}
Title: \{subject\}
Field: \{area, major, program\}
Research-field: \{Research-area\}
Requirement: \{qualification\}
Admission-date: \{date, deadline, start-date, end-date\}
Amount: \{fee\}
Address: \{Permanent-address, Postal-address, Add.\}
Publish-date: \{Pub-date, date, year\}
Author: \{writer, Published-by, Publisher\}
Position: \{post, job, Job-title\}
Event-date: \{start-date, date, deadline,important-date\}
Description: \{Description\}
Project-Title: \{title, subject\}
Language: ()

The synonym sets of relationships:

Advisor: \{supervisor\}
Org-Publication: \{organization-publication\}
Research-Publication: ()
Project-Publication: ()
Phase 7. Modeling: In this step we design a concept hierarchy graph and a relation concept graph by one ontology-editing tool. In this phase, we use Protégé-2000 for our example (shown in Figure 6.3). Protégé-2000 was developed by Mark Musen’s group at Stanford Medical Informatics.
Phase 8: Implementation. Finally we implement and store the uniform representation models (hierarchy concept and relation concept graphs) of the ontology under one DBMS such as SQL/SERVER, MySQL or Access (Figure 6.4).

6.3 Graphic User Interface

We implement GUI to represent the university domain ontology through ASP.NET in visual studio environment. Figures 6.5 and 6.6 show the interfaces for the ‘University’ and ‘Staff’ concepts from the university domain ontology.
6.4 Sample Local Ontologies

We use USM (University Science Malaysia) and UTM (University Technology Malaysia) as sample local ontologies for evaluation of our mapping algorithm. Figures 6.7 and 6.8 show the concept hierarchy graph of USM ontology and UTM ontology.
Fig. 6.7 Concept hierarchy diagram of USM ontology
Fig. 6.8 Concept hierarchy diagram of UTM ontology
USM and UTM ontologies are extracted from the University Science Malaysia and University Technology Malaysia web sites. The USM ontology consists of 98 concepts, 20 attributes and 6 relationships. The UTM ontology consists of 114 concepts, 23 attributes and 9 relationships.

### 6.5 Implementation and Assessment of Mapping Algorithm

Our mapping algorithm (mentioned in Chapter 5) was implemented in .NET framework. The algorithm was experimented with three manually constructed ontologies: The university domain ontology, which has 104 concepts, 20 different attributes and 10 relationships (explained in Section 6.2); and USM (University Science Malaysia) ontology, which has 98 concepts, 20 attributes and 6 relationships and UTM (University Technology Malaysia) ontology, which has 114 concepts, 23 attributes and 9 relationships (please refer to Section 6.4). University domain ontology is a domain specific ontology in area of university, which was developed for our system. The quality of mapping results has direct relation with completeness of domain ontology. If we enhance completeness and accuracy of our university domain ontology, we will have higher quality mapping results. USM and UTM ontologies are sample local ontologies, which were extracted from University Science of Malaysia and University Technology Malaysia websites.

**First Experimental results:** Table 6.1 shows experimental results of our mapping algorithm executed on university domain and USM ontologies. We posed 40 queries through the browsing the university domain ontology. There exists totally 128 query terms in all posed queries. Afterwards, we executed mapping algorithm on each query to find the
corresponding mapping terms from the USM ontology. The quality measurements that represent measure for quality of mapping results are as follows (Melnik et al., 2002):

- **P** is the number of user queries: It shows the number of all queries that have been posed by users.

- **Q** is the number of user queries terms: It shows the number of all terms that exist in the above mentioned queries.

- **R** is the number of query mappings returned by the algorithm: It shows the number of mapped queries that the algorithm returns as a successful query mapping. For example in our experiment, with 0.6 as a threshold value, 21 mapped queries from 40 posed queries were returned by the algorithm.

- **S** is the number of terms mappings returned by the algorithm: It shows the number of all query terms that have been mapped to the local ontology terms. For example, in our experiment, 52 query terms from 128 query terms have been mapped to the USM ontology terms by the algorithm (with threshold 0.6).

- **R_2** is the number of manually determined real query mappings; for example in our experiment, 24 queries from 40 posed queries, manually have been mapped successfully to the USM ontology. These 24 mapped queries are real query mappings.

- **T** is the number of manually determined real terms mappings; for example in our experiment, 60 query terms from 128 posed query terms, manually have been mapped successfully to the USM ontology terms. These 60 mapped terms are real query terms mappings.

- **U** is the number of correctly identified mappings (true mapping terms); for example in our experiment, 43 mapping terms from 52 mapped query terms have been mapped successfully and are correct mappings (with threshold 0.6).
• $V = S \setminus U$ is the number of false mappings; for example in our experiment, 9 mapping terms from 52 mapped query terms have been mapped wrongly and are incorrect mappings (with threshold 0.6).

• $W = T \setminus U$ is the number of missing mappings; for example in our experiment, 17 real mapping terms from 60 existing real mapping terms between two ontologies have been missed by the algorithm (with threshold 0.6).

• Precision = $|U| / |S|$ estimate the reliability of the mapping predictions: It represents a mapping quality measurement. It is a value in the range of $[0,1]$. Higher value represents higher quality.

• Recall = $|U| / |T|$ specifies the share of real mappings found: It represents a mapping quality measurement. It is a value in the range of $[0,1]$. Higher value represents higher quality and

• Overall = Recall* (2 – 1/Precision) is a combined mapping quality measurement in the value range of $[0,1]$.

Table 6.1 Mapping algorithm results on the University and USM ontologies

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>Threshold</th>
<th>$R$</th>
<th>$S$</th>
<th>$R2$</th>
<th>$T$</th>
<th>$U$</th>
<th>$V = S \setminus U$</th>
<th>$W = T \setminus U$</th>
<th>Precision</th>
<th>Recall</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>128</td>
<td>0.6</td>
<td>21</td>
<td>52</td>
<td>24</td>
<td>60</td>
<td>43</td>
<td>9</td>
<td>17</td>
<td>.8269</td>
<td>.7166</td>
<td>.5665</td>
</tr>
<tr>
<td>40</td>
<td>128</td>
<td>0.7</td>
<td>18</td>
<td>45</td>
<td>24</td>
<td>60</td>
<td>38</td>
<td>7</td>
<td>23</td>
<td>.8409</td>
<td>.6065</td>
<td>.4917</td>
</tr>
<tr>
<td>40</td>
<td>128</td>
<td>0.8</td>
<td>17</td>
<td>39</td>
<td>24</td>
<td>60</td>
<td>34</td>
<td>5</td>
<td>27</td>
<td>.8717</td>
<td>.5573</td>
<td>.4752</td>
</tr>
<tr>
<td>40</td>
<td>128</td>
<td>0.9</td>
<td>14</td>
<td>31</td>
<td>24</td>
<td>60</td>
<td>28</td>
<td>3</td>
<td>33</td>
<td>.9032</td>
<td>.4590</td>
<td>.4098</td>
</tr>
<tr>
<td>40</td>
<td>128</td>
<td>1.00</td>
<td>14</td>
<td>29</td>
<td>24</td>
<td>60</td>
<td>27</td>
<td>2</td>
<td>34</td>
<td>.9310</td>
<td>.4426</td>
<td>.4097</td>
</tr>
</tbody>
</table>

The diagrams in Figure 6.9, 6.10, 6.11 and 6.12 are related to the data of table 6.1 and represent experimental results of our mapping algorithm. Figure 6.9 shows the number of mapped queries decrease if the threshold increases (shown in Figure 6.9 with yellow line) because if we increase the threshold of the mapping function then two terms must be further similar in name till they totally correspond to each other.
Fig. 6.9 Mapping quality measure diagram 1

Figure 6.10 shows the number of mapping terms (S) and correct mapping terms (U) decrease if the threshold increases. As mentioned before if we increase threshold then two terms must be further similar in name and string till be mapped to each other. For this reason, the number of mapping terms and true mapping terms become farther from manually real mapping terms (T) by increasing of threshold.
Figure 6.11 shows the number of false mapping terms (V) decrease if the threshold increases. This is natural since if we increase the threshold then the quality of the mapping increases and subsequently the number of false mappings decrease. The number of missing mapping terms (W) increases if we increase the threshold because while we increase threshold of mapping function then the algorithm just finds those terms that are further similar in name and string to each other and misses some real semantically similar terms.

![Mapping quality measure diagram 3](image)

Fig 6.11 Mapping quality measure diagram 3

Precision, recall and overall are measurements for mapping quality with value in the range of [0,1]. If all returned mapping results are the same with the real results then precision, recall and overall would be 1. Figure 6.12 shows while we increase the threshold, precision increases but recall with overall decreases. Our expectation and goal for mapping quality of our algorithm is a value above 0.7 for average of precision, recall and overall. The diagram in Figure 8.12 shows we attain our goal with a threshold of 0.6 because precision is 0.8269, recall is 0.7166 and overall is 0.5665 (average is 0.7033).
Second Experimental results: Table 6.2 shows experimental results of our mapping algorithm executed on the university domain and UTM ontologies. We used the same queries posed in our first experimental. We executed the mapping algorithm on each query to find corresponding mapping terms from the UTM ontology.

Table 6.2 Mapping algorithm results on University and UTM ontologies

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>Threshold</th>
<th>R</th>
<th>S</th>
<th>R2</th>
<th>T</th>
<th>U</th>
<th>S\U</th>
<th>V = S\U</th>
<th>W = T\U</th>
<th>Precision</th>
<th>Recall</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>131</td>
<td>0.6</td>
<td>23</td>
<td>58</td>
<td>29</td>
<td>71</td>
<td>50</td>
<td>8</td>
<td>21</td>
<td>.8620</td>
<td>.7042</td>
<td>.5914</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>131</td>
<td>0.7</td>
<td>19</td>
<td>48</td>
<td>29</td>
<td>71</td>
<td>41</td>
<td>7</td>
<td>30</td>
<td>.8541</td>
<td>.5774</td>
<td>.4787</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>131</td>
<td>0.8</td>
<td>17</td>
<td>41</td>
<td>29</td>
<td>71</td>
<td>36</td>
<td>5</td>
<td>35</td>
<td>.8780</td>
<td>.5070</td>
<td>.4365</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>131</td>
<td>0.9</td>
<td>14</td>
<td>33</td>
<td>29</td>
<td>71</td>
<td>29</td>
<td>4</td>
<td>42</td>
<td>.8787</td>
<td>.4084</td>
<td>.3521</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>131</td>
<td>1.00</td>
<td>14</td>
<td>31</td>
<td>29</td>
<td>71</td>
<td>28</td>
<td>3</td>
<td>43</td>
<td>.9032</td>
<td>.3943</td>
<td>.3520</td>
<td></td>
</tr>
</tbody>
</table>

The diagrams in figure 6.13, 6.14, 6.15 and 6.16 are related to data of table 6.2 and represent experimental results for our mapping algorithm. Figure 6.13 shows the number of mapped queries decrease if the threshold increases (shown in figure 6.13 with yellow line).
Figure 6.13 Mapping quality measure diagram 5

Figure 6.14 shows number of mapping terms (S) and correct mapping terms (U) decrease if the threshold increases.

Figure 6.14 Mapping quality measure diagram 6

Figure 6.15 shows number of false mapping terms (V) decrease if threshold increases. The number of missing mapping terms (W) increases if we increase threshold.
Fig. 6.15 Mapping quality measure diagram 7

Figure 6.16 shows while we increase the threshold, precision increases but recall with overall decreases. Our expectation and goal for mapping quality of our algorithm is a value above 0.7 for average of precision, recall and overall. The diagram in figure 6.16 shows that we attain our goal with a threshold of 0.6 because precision is 0.8620, recall is 0.7042 and overall is 0.5914 (average is 0.7192).

Fig. 6.16 Mapping quality measure diagram 8
6.6 Conclusion

In this chapter, we focused on the implementation of our ontology mapping approach. For this purpose, we presented a methodology in order to build the domain ontologies based on proposed uniform representation model and created a sample university domain ontology. We presented USM and UTM ontologies as two sample local ontologies in order to evaluate our mapping algorithm. We implemented proposed mapping algorithm in .NET framework and experimented with mentioned ontologies: university domain ontology, USM ontology and UTM ontology. Our expectation and goal for mapping quality of our algorithm is a value above 0.7 for average of precision, recall and overall. The quality of mapping results has direct relation with completeness of domain ontology. If we enhance completeness and accuracy of our university domain ontology, we will have higher quality mapping results.

The diagrams in section 6.5 show the quality of mapping results based on our created university domain ontology. The diagrams in figure 6.12 and 6.16 show we attain our goal with a threshold of 0.6 because in the first experiment precision is 0.8333, recall is 0.7377 and overall is 0.5901 (average is 0.7203) and second experiment precision is 0.8620, recall is 0.7042 and overall is 0.5914 (average is 0.7192).
Chapter 7: Conclusion and Discussion
Chapter 7 Conclusion and Discussion

7.1 Introduction

This chapter concludes the thesis. Section 7.2 presents the summary of research and Section 7.3 states research results and contributions. Finally, we introduce ideas for further research in Section 7.4.

7.2 Summary of the Research

The main goal of this research is to “present an approach for resolving semantic heterogeneities at schema level in the integration of web sources”. In order to achieve this goal, we focused over two major works as follows:

1. Propose an approach and system architecture for web data integration system: We proposed an approach and system architecture for integration of heterogeneous web data sources (explained in Chapter 4). The proposed web data integration system relies on the ontology technology for resolving of semantic schema heterogeneity.

2. Propose an algorithm for semantic mapping between ontologies: In the second step, we focused over semantic mapping component of the proposed web data integration system and proposed a semantic mapping algorithm for reconciliation of semantic conflicts between web ontologies. This algorithm resolves semantic schema heterogeneity between web data sources (presented in Chapter 5).

As explained in Chapter 4, the proposed web data integration approach uses ontologies for resolving of semantic conflicts between web data sources. The proposed approach uses
domain specific ontologies for the creation of user queries. There is a domain specific ontology for each application that covers the semantic definition of terms which are required for user queries in a particular application domain. The domain ontologies are modeled in a uniform representation model. The user can browse the domain ontology and choose terms for his/her query; afterwards the system creates the user query. We assume each web source has an underlying pre-existing local ontology on the web and each local ontology is associated with one or more web sources. After the creation of the user query, the web ontology server chooses the related local ontologies which is relevant to the user query domain and sends them to the mapping module. First, the local ontology is transformed to the system uniform representation model and then the user query terms are mapped to the corresponding terms from the local ontology. Subsequently, the user query is rewritten using the corresponding terms of the local ontology. This system uses ontologies to resolve semantic schema conflicts between web data sources. The proposed system resolves semantic schema heterogeneities between web source and user query through semantic mapping between the domain ontology and the local ontology.

The rewritten user query is sent to a query process module for reformulation and creation of optimized query plan from the user query. The sub queries from the reformulation process are translated to the web sources query languages by translators and then the answer to the sub-queries are extracted from related web sources by wrappers. Each wrapper knows the structure of the underlying web source and extracts the related data to its sub-query and presents the extracted data in a common format. Through wrappers, the system resolves data model heterogeneity conflicts between web data sources.

The extracted data obtained through wrappers are sent to the converter. The converter resolves data value heterogeneity conflicts between data values. The converter exploits conversion functions and rules for resolving heterogeneities between two data value (such
as units, data types, value format conversion functions). Finally the query plan operations
are executed and the final answers are exported to the user’s preference format by the
exporter.

Our proposed approach is a domain specific approach because the user is confined to
choosing terms of a specific domain ontology for his/her query. This approach can be
extended for any domain so that the relevant domain ontology would be previously
developed. For example in order to use this approach in hospital domain we need to create
and develop one hospital domain ontology and use in system as domain ontology. The
extraction and integration process in the proposed system consists of eleven major tasks as
follows:

1. Creation of user query;
2. Determination of related local ontologies and their underlying web sources with query
domain;
3. Transformation of related local ontologies to internal uniform representation model;
4. Semantic mapping between query terms and related local ontologies terms;
5. Rewriting of user query with corresponding terms from local ontologies;
6. Reformulation of query and creation of optimized query plan;
7. Translation of sub queries to web wrapper query languages;
8. Extraction of data;
9. Conversion of data values;
10. Execution of query plan operations;
11. Exporting of data;

The software architecture of the proposed web data integration system consists of six
modules as follows:

1. query construction module
2. web ontology server
3. semantic mapping module
4. query process module
5. data extraction module and
6. data correlation module

Based on the purpose of second work, the research focus on query construction module and semantic mapping module and proposed approaches for implementation of these two modules. We suggested an approach for the creation of user query and an algorithm for resolving semantic schema conflicts between user query terms (chosen from domain ontology) and related local ontologies terms. Our work covers the first, third, fourth and fifth tasks of the integration process mentioned above.

In our work we proposed one uniform representation model for ontologies. This representation model is general and any ontology with any representation model can be transformed to this uniform representation model. The mapping algorithm works base on features and definition mechanisms of this representation model.

In proposed query construction approach, the user traverses the internal uniform representation graphs of the domain ontology in order to choose his/her query terms. The path through which a user traverses in the domain ontology graphs for reaching his/her query terms is called the query path. We use this query path as comparison and mapping criteria for finding similar and corresponding terms between the user query and the local ontologies in the semantic mapping algorithm.

The user is confined to use just one domain ontology for his/her query. Users cannot pose complex queries because the query construct and structure is based on the system query structure and based on elements of the uniform representation model of the domain
ontology. We defined the following structure and syntax as system query structure and syntax for the expression of the user query.

\[
\begin{align*}
\text{SELECT} & \quad < \text{attributes names} > \\
\text{FROM} & \quad < \text{concept name} > \\
\text{WHERE} & \quad \{ <\text{attribute names: values}> \text{ FROM } <\text{concept name}_1> \\
& \quad \quad <\text{attribute names: values}> \text{ FROM } <\text{concept name}_2> \ldots \}
\end{align*}
\]

In this query structure, the user can query the attributes of only one concept from the domain ontology (we call it concept query). That means each user query possesses only one concept query. The user can specify constraints and conditions on his/her query. Constraints and conditions are expressed after the “WHERE” clause in the query expression.

A user interacts with the system through the GUI (Graphical User Interface) in the query construction module of the system. The GUI must display the domain ontology to the user so that the user can find his/her query terms easily and quickly. In the time the user takes to interact with the system for the construction of the query, the system performs two tasks: first creation of query and second specifying of query path for use in the semantic mapping algorithm.

As explained in Chapter 5 the proposed mapping algorithm consists of eight steps. Within these eight steps, algorithm finds corresponding terms with query path terms from local ontology and maps them to each other. Inputs of mapping algorithm are: query path, domain ontology and local ontology. There are three types of ontology element (term) in a query path: concept (C), attribute (A) and relationship (R). The query path consists of the user query terms and some un-query terms. The un-query terms are terms which have been traversed by user for reaching to query terms. These terms do not exist in the user query.

The purpose of mapping algorithm is to find local ontology terms that are semantically similar with the terms in the query path and then rewrite the user query with the
corresponding similar terms. The table 7.1 shows specifications of proposed mapping algorithm.

Table 7.1 specification of proposed mapping algorithm

<table>
<thead>
<tr>
<th>Mapping function specification</th>
<th>Our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of mapping function</td>
<td>Multiple</td>
</tr>
<tr>
<td>Number of mapping criterion</td>
<td>Multiple</td>
</tr>
<tr>
<td>Combination of mapping function</td>
<td>Composition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping algorithm technique</th>
<th>Mapping technique</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linguistic-based</td>
</tr>
<tr>
<td></td>
<td>Constraint-based</td>
</tr>
<tr>
<td></td>
<td>Path-based</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping granularity</th>
<th>Element-level</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Mapping element</th>
<th>Concept, instance, attribute</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Mapping cardinality</th>
<th>1:1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Input &amp; output</th>
<th>Ontology representation model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Graph stored in relational data-based</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Auxiliary information</th>
<th>domain specific dictionary</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Input of algorithm</th>
<th>User query path, domain ontology, local ontology</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Output of algorithm</th>
<th>Concept mapping table, attribute mapping table</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>User interaction</th>
<th>Selection of query and query path terms, approve or disapprove of answer</th>
</tr>
</thead>
</table>

| Complexity of algorithm       | n log n                                                                  |

As mentioned in sub-section 5.6.2 the ontology mismatches are categorized in the language level and the ontology level. The ontology level mismatches consist of the domain of description, Terminology, Paradigm and description model mismatches. Our mapping algorithm resolves:

- Language level mismatches by transforming ontologies to the system internal uniform representation model.
- Domain of description mismatches by synonym set matching. Step 6 of mapping algorithm also helps to resolve domain of description mismatches.
- Terminology mismatches by the synonym set matching, key property matching and father matching in mapping algorithm.
• Paradigm mismatches by synonym set matching. Key property matching and father matching in mapping algorithm also helps to resolve paradigm mismatches.

• Description model mismatches by the exceptional tasks 1 and 2 (step 7 in the algorithm).

In next section, we explain further about results of our proposed mapping algorithm.

7.3 Results and Contributions of the Research

The purpose of the work is to introduce an approach for resolving semantic schema heterogeneities in web data integration. The overall purpose is decomposed into the intermediate goals of this work. The goals of this work are to:

• Clarify and define semantic schema conflicts (semantic heterogeneities at the schema level). We dealt with this issue in Chapter 3. We defined semantic schema is related to the meaning of schema elements (e.g., classes, attributes and relationships). Any user or application has its own semantics based on requirements and common sense from the application domain. We refer to different interpretations of data as semantic heterogeneity and if such differences are related to schema elements we refer to those as semantic schema heterogeneity (Geppert and Hakimpour, 2001).

• Formalization of semantics through ontology. We explored the required formalism mechanisms for an ontology language and introduce OWL as new developed ontology language for formalization of web ontologies in Chapter 3.

• Propose a web data integration system. We proposed an approach and system architecture for integration of web data sources. We presented this system in Chapter 4.
• Propose an algorithm for semantic mapping between web ontologies: This algorithm finds semantic corresponding terms among the web ontologies and maps them to each other. We presented this algorithm in Chapter 5.

We discuss our contributions according to these objectives. In this thesis, the overall problem of semantic schema conflicts is resolved and approached with focus on semantic mapping between domain ontology (query path terms) and local ontology. The main contributions of this work are as follows:

1. **Propose system architecture for web data integration:** The proposed approach uses domain specific ontologies for the creation of user queries. There is a domain specific ontology for each application domain that covers all the semantic definition of existing terms in a particular application domain. Users can browse the domain ontology and choose terms and concepts for his/her query, the system then creates a user query based on the system query language. We assume each web source has one or more underlying pre-existing local ontology and each local ontology is associated with one or more web sources. User query terms are mapped to the local ontology terms and the user query is subsequently rewritten using the terms from the local ontology. Finally, the user query is translated to the web source query language and the data is extracted, converted and presented to the user.

Our proposed web data integration system covers all abstraction levels of data heterogeneity conflicts between web data sources. The system applies:

- ontology as a solution for resolving schema heterogeneities;
- wrapper as solution for resolving data model heterogeneities;
- converter as solution for resolving data value heterogeneities;
The proposed web data integration system is scalable to any domain by adding a related domain ontology to the system. That means in order to use the system in any application area we must develop and add a domain ontology (relevant to the application domain area) to the system.

2. **Propose ontology mapping algorithm and implement a prototype to evaluate proposed algorithm:** The following features are advantage points of proposed algorithm. We used the following techniques and features for enhancement of the mapping quality and help to resolve mismatches between ontologies in proposed algorithm:

- **Posing user query based on domain ontology terms:** This resolves semantic conflicts in presentation layer of system (user level) because different users use same terms for their queries and they possess same view of domain and application area.

- **Modeling ontologies based on Uniform representation model:** The proposed uniform representation model is general and possesses main exploitable information in an ontology. The existing definition mechanisms in uniform representation model help to resolve mismatches. The synonym set and key property features in proposed uniform representation model help to enhance mapping result quality. As mentioned in the previous section, these features help to resolve domain of description, terminology and paradigm mismatches between ontologies.

- **Using query path in mapping algorithm:** The query path is a most effective mapping technique of the proposed algorithm. The algorithm searches the domain ontology in order to find the user query concept and its attributes through the query path. The query path provides two advantages:
1. **Reduce runtime of each achievement of mapping results:** It directs algorithm toward query concept and its attributes. It causes to reduce the search domain of algorithm. As calculated in Section 5.8 the runtime of algorithm possesses complexity of $O(n \log n)$.

2. **Gain higher quality mapping results:** The query path possesses entities (concepts) which have some semantical relation with the query concept because they exist on a same path of the ontology graph. Terms on a same path are related to each other through is-a or R relationship. Therefore the algorithm has further information about the semantic of the query concept that helps to find corresponding terms with the query concept from the local ontology. Hence the query path causes higher quality mapping results.

---

### 7.4 Future Works

Web data integration is a complex process and need a complete automated system to resolve all heterogeneity levels between web data. The proposed web data integration system tries to resolve all heterogeneity levels. The software architecture of the proposed web data integration system consists of six modules as follows (Chapter 4, Figure 4.6):

1. query construction module
2. web ontology server module
3. semantic mapping module
4. query process module
5. data extraction module and
6. data correlation module
We need to have approach and algorithm in order to implement all the above mentioned modules. As future work we need to propose approaches for implementation of other modules of proposed web data integration system as follows:

- In order to implement web ontology server module we need to have an approach for storing and maintaining of required Information about relevant local ontologies and the data sources underlying each local ontology residing on the web. The web ontology server should provide the following services:
  - Obtaining existing terms from the ontology
  - Obtaining definitions of terms
  - Graphical representation of the ontology
  - Selecting the local ontologies that are related to the domain of the user’s query

- The query process module is complex module in web data integration system. The approach for implementation of query process module should cover the following main tasks:
  - *Reformulation of user query* into one or more sub queries based on the content of the local ontologies and web sources.
  - *Creation and optimization of the query plan* in order to find the optimal query execution plan for these sub queries.

- In the extraction module, we need to implement wrapper for web sources. A wrapper is a module which understands a specific data organization. It knows how to retrieve data from the underlying repository.

- The main task in data correlation module is conversion of data values. In order to implement this module we need to have a conversion algorithm. That would resolve the
data value heterogeneity conflicts between extracted data values. The converter exploits conversion functions and rules for resolving heterogeneities between two data value.

The implementing all above mentioned modules and developing whole proposed web data integration system needs to a complete research group and enough time. In this research based on research’s goals and time constraint, we explored the query construction and semantic mapping modules and proposed approaches for implementation of these two modules.
APPENDIX
Appendix A: OWL Coding of University Ontology

This appendix contains some part of OWL code of the domain specific University ontology.

```xml
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [ 
<!ENTITY owl "http://www.w3.org/2002/07/owl#" >
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY p1 "http://www.owl-ontologies.com/assert.owl#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]> 
<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1193423774.owl#"
xml:base="http://www.owl-ontologies.com/Ontology1193423774.owl"
xmlns:p1="http://www.owl-ontologies.com/assert.owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#">
<owl:Ontology rdf:about=""/>
<owl:Class rdf:ID="ABOUT_COLLEGE">
<rdfs:subClassOf rdf:resource="#UNIVERSITY"/>
<owl:disjointWith rdf:resource="#STUDENT"/>
<owl:disjointWith rdf:resource="#FINANCIAL_AID"/>
<owl:disjointWith rdf:resource="#EVENT"/>
<owl:disjointWith rdf:resource="#COLLEGE"/>
<owl:disjointWith rdf:resource="#PROJECT"/>
<owl:disjointWith rdf:resource="#ADMISSION"/>
<owl:disjointWith rdf:resource="#RESEARCH-GROUP"/>
<owl:disjointWith rdf:resource="#TUITION_FEE"/>
<owl:disjointWith rdf:resource="#HISTORY"/>
<owl:disjointWith rdf:resource="#DEAN_MESSAGE"/>
<owl:disjointWith rdf:resource="#INSTITUTE"/>
<owl:disjointWith rdf:resource="#PROGRAM"/>
<owl:disjointWith rdf:resource="#LIBRARY"/>
<owl:disjointWith rdf:resource="#LOCATION_MAP"/>
<owl:disjointWith rdf:resource="#CONTACT_INFORMATION"/>
<owl:disjointWith rdf:resource="#CENTER"/>
<owl:disjointWith rdf:resource="#STAFF"/>
<owl:disjointWith rdf:resource="#SUBJECT"/>
<owl:disjointWith rdf:resource="#FACILITY"/>
<owl:disjointWith rdf:resource="#CAREER"/>
<owl:disjointWith rdf:resource="#PUBLICAITION"/>
</owl:Class>
<owl:Class rdf:ID="ACADEMIC-DEPUTY-DEAN">
<rdfs:subClassOf rdf:resource="#UNIVERCITY-DEPUTY-DEAN"/>
<owl:disjointWith rdf:resource="#DEVELOPMENT-DEPUTY-DEAN"/>
</owl:Class>
<owl:Class rdf:ID="ACADEMIC-STAFF">
<rdfs:subClassOf rdf:resource="#STAFF"/>
<owl:disjointWith rdf:resource="#ADMINISTRATIVE-STAFF"/>
</owl:Class>
<owl:Class rdf:ID="ACADEMIC_POSITION">
<rdfs:subClassOf rdf:resource="#CAREER"/>
<owl:disjointWith rdf:resource="#NON_ACADEMIC_POSITION"/>
</owl:Class>
```
Appendix B: OWL Coding of USM Ontology

This appendix contains some part of OWL code of the USM local ontology.

```xml
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
<!ENTITY owl "http://www.w3.org/2002/07/owl#" >
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]> 

<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1205113384.owl#" 
xml:base="http://www.owl-ontologies.com/Ontology1205113384.owl" 
xmlns:xsd="http://www.w3.org/2001/XMLSchema#" 
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" 
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" 
xmlns:owl="http://www.w3.org/2002/07/owl#"> 
<br:Ontology rdf:about=""/> 
<br:Class rdf:ID="ABOUT_USM"> 
  <rdfs:subClassOf rdf:resource="#UNIVERSITY"/> 
</br:Class> 
<br:Class rdf:ID="ACADEMIC_CALENDAR"> 
  <rdfs:subClassOf rdf:resource="#UNIVERSITY"/> 
</br:Class> 
<br:Class rdf:ID="ACADEMIC_CENTRE"> 
  <rdfs:subClassOf rdf:resource="#CENTRE"/> 
</br:Class> 
<br:Class rdf:ID="ACADEMIC_PROGRAM"> 
  <rdfs:subClassOf rdf:resource="#FULLTIME_DEGREE"/> 
</br:Class> 
<br:Class rdf:ID="ACADEMIC_PROGRAMME"> 
  <rdfs:subClassOf rdf:resource="#SCHOOL"/> 
</br:Class> 
<br:Class rdf:ID="ACADEMIC_STAFF"> 
  <rdfs:subClassOf rdf:resource="#STAFF"/> 
</br:Class> 
<br:Class rdf:ID="ACADEMY_CALENDAR"> 
  <rdfs:subClassOf rdf:resource="#ADMISSION"/> 
</br:Class> 
<br:Class rdf:ID="ADMIN_STAFF"> 
  <rdfs:subClassOf rdf:resource="#STAFF"/> 
</br:Class> 
<br:Class rdf:ID="ADMINISTRATIVE_CENTRE"> 
  <rdfs:subClassOf rdf:resource="#CENTRE"/> 
</br:Class> 
<br:Class rdf:ID="ADMINISTRATIVE_STAFF"> 
  <rdfs:subClassOf rdf:resource="#ORGANISATION"/> 
</br:Class> 
<br:Class rdf:ID="ADMISSION"> 
  <rdfs:subClassOf rdf:resource="#UNIVERSITY"/> 
</br:Class> 
<br:Class rdf:ID="ADMISSION_PROCEDURE"> 
  <rdfs:subClassOf rdf:resource="#ADMISSION"/> 
</br:Class>
```
<owl:Class rdf:ID="DOWNLOAD_FORM">
    <rdfs:subClassOf rdf:resource="#INSTITUTE_GRADUATESTUDY"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="download_pdf">
    <rdfs:domain>
        <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#DOWNLOAD_FORM"/>
            <owl:Class rdf:about="#FEE_STRUCTURE"/>
            <owl:Class rdf:about="#POSTDOCTORAL_FELLOWSHIP"/>
        </owl:unionOf>
    </rdfs:domain>
</owl:ObjectProperty>

<owl:Class rdf:ID="E_LEARNING">
    <rdfs:subClassOf rdf:resource="#UNIVERSITY"/>
</owl:Class>

<owl:Class rdf:ID="E_RECRUIT">
    <rdfs:subClassOf rdf:resource="#STAFF_RECRUITEMENT"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="e_recruit">
    <rdfs:domain>
        <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#ACADEMIC_STAFF"/>
            <owl:Class rdf:about="#ADMIN_STAFF"/>
            <owl:Class rdf:about="#ADMINISTRATIVE_STAFF"/>
            <owl:Class rdf:about="#Prof"/>
            <owl:Class rdf:about="#SUPPORT_STAFF"/>
            <owl:Class rdf:about="#TECHNICAL_STAFF"/>
        </owl:unionOf>
    </rdfs:domain>
</owl:ObjectProperty>

<owl:Class rdf:ID="ENTRANCE_REQUIREMENT">
    <rdfs:subClassOf rdf:resource="#ADMISSION"/>
</owl:Class>

<owl:Class rdf:ID="FACILITY">
    <rdfs:subClassOf rdf:resource="#ADMISSION"/>
</owl:Class>

<owl:Class rdf:ID="FEE">
    <rdfs:subClassOf rdf:resource="#ADMISSION"/>
</owl:Class>

<owl:Class rdf:ID="FEE_STRUCTURE">
    <rdfs:subClassOf rdf:resource="#INSTITUTE_GRADUATESTUDY"/>
</owl:Class>

<owl:Class rdf:ID="FINANCIAL_ASSISTANCE">
    <rdfs:subClassOf rdf:resource="#INSTITUTE_GRADUATESTUDY"/>
</owl:Class>

<owl:Class rdf:ID="FUULTIME_DEGREE">
    <rdfs:subClassOf rdf:resource="#ADMISSION"/>
</owl:Class>

<owl:Class rdf:ID="GOVERNANCE">
    <rdfs:subClassOf rdf:resource="#ABOUT_USM"/>
</owl:Class>
<owl:Class rdf:ID="GRADUATESTUDY_INSTITUTE"/>
<owl:Class rdf:ID="HISTORY"/>
<owl:Class rdf:ID="HOSTEL"/>
<owl:Class rdf:ID="INSTITUTE_GRADUATESTUDY"/>
<owl:Class rdf:ID="INSTITUTE_POSTGRADUATE"/>
<owl:Class rdf:ID="INTERNATIONAL_APPLICANT"/>
<owl:Class rdf:ID="INTERNATIONAL_STUDNET"/>
<owl:Class rdf:ID="ISLAMIC_CENTRE"/>
<owl:Class rdf:ID="KNOWLEDGE_CENTRE"/>
<owl:Class rdf:ID="LANGUAGE_CENTRE"/>
<owl:Class rdf:ID="LANGUAGE_REQUIREMENT"/>
<owl:Class rdf:ID="LIBRARY"/>
<owl:Class rdf:ID="LOGO"/>
<owl:Class rdf:ID="MALAYSIAN_APPLICANT"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
<owl:ObjectProperty rdf:ID="member"/>
<owl:ObjectProperty rdf:ID="grant_name"/>
Appendix C: OWL Coding of UTM Ontology

This appendix contains some part of OWL code of the UTM local ontology.

```xml
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [ 
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" > 
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" > 
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" > 
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" > 
]>

<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1205113384.owl#" 
  xml:base="http://www.owl-ontologies.com/Ontology1205113384.owl" 
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#" 
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" 
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" 
  xmlns:owl="http://www.w3.org/2002/07/owl#">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="ABOUT_UNIVERSITY">
    <rdfs:subClassOf rdf:resource="#UNIVERSITY"/>
  </owl:Class>
  <owl:Class rdf:ID="ACADEMIC_CALENDAR">
    <rdfs:subClassOf rdf:resource="#ACADEMIC_PROGRAMME"/>
  </owl:Class>
  <owl:Class rdf:ID="ACADEMIC_CALENDARS">
    <rdfs:subClassOf rdf:resource="#UNDERGRADUATE_ADMISSION"/>
  </owl:Class>
  <owl:Class rdf:ID="ACADEMIC_DEVELOPMENT_SUPPORT">
    <rdfs:subClassOf rdf:resource="#STAFF"/>
  </owl:Class>
  <owl:Class rdf:ID="ACADEMIC_PROGRAMME">
    <rdfs:subClassOf rdf:resource="#UNIVERSITY"/>
  </owl:Class>
  <owl:Class rdf:ID="ACADEMICS_CALENDAR">
    <rdfs:subClassOf rdf:resource="#POSTGRADUATE_ADMISSION"/>
  </owl:Class>
  <owl:Class rdf:ID="ACCOMODATION">
    <rdfs:subClassOf rdf:resource="#CURRENT_STUDENT"/>
  </owl:Class>
  <owl:Class rdf:ID="ACHIEVEMENT_AWARDS">
    <rdfs:subClassOf rdf:resource="#UNIVERSITY"/>
  </owl:Class>
  <owl:Class rdf:ID="ADMINISTRATIVE_UNIT">
    <rdfs:subClassOf rdf:resource="#ABOUT_UNIVERSITY"/>
  </owl:Class>
  <owl:Class rdf:ID="ADMISSION">
    <rdfs:subClassOf rdf:resource="#UNIVERSITY"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="APPLICATION_FORM"/>
  <owl:Class rdf:ID="APPLICATION_FORMS">
    <rdfs:subClassOf rdf:resource="#INTERNATIONAL_STUDENT"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="award_name"/>
</rdf:RDF>
```
Appendix D: Implementation of Graphic User Interface

This appendix contains partially implementation code of the GUI for presentation of university ontology to user. We use ASP.net for coding of GUI web pages.

Partial Class _Default
Inherits System.Web.UI.Page

    Protected Sub CheckBox1_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
        End Sub

    Protected Sub TextBox2_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles txtboxName.TextChanged
        Dim connectionString As String = "provider=Microsoft.jet.oledb.4.0;" & "data source=D:\university\university.mdb"
        Dim dbConnection As New Data.OleDb.OleDbConnection
dbConnection.ConnectionString = connectionString
        dbConnection.Open()

        Dim commandString As String = "INSERT INTO attribute_querypath(Concept_name, Attribute_name, Value)" & "values(@Concept_name, @Attribute_name, @Value)"
        Dim dbCommand As New Data.OleDb.OleDbCommand(commandString, dbConnection)
        Dim conceptNameParam As New Data.OleDb.OleDbParameter
        Dim attributeNameParam As New Data.OleDb.OleDbParameter
        Dim valueParam As New Data.OleDb.OleDbParameter

        conceptNameParam.ParameterName = "@Concept_name"
        attributeNameParam.ParameterName = "@Attribute_name"
        valueParam.ParameterName = "@Value"

        conceptNameParam.Value = "Full_professor"
        attributeNameParam.DbType = Data.DbType.String
        attributeNameParam.Value = LblName.Text
        valueParam.Value = txtboxName.Text
        valueParam.DbType = Data.DbType.String

        dbCommand.Parameters.Add(valueParam)
dbCommand.ExecuteNonQuery()
        dbConnection.Close()
    End Sub

    Protected Sub TextBox3_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles txtboxEmail.TextChanged
        Dim connectionString As String = "provider=Microsoft.jet.oledb.4.0;" & "data source=D:\university\university.mdb"
        Dim dbConnection As New Data.OleDb.OleDbConnection
    End Sub
dbConnection.ConnectionString = connectionString
dbConnection.Open()

Dim commandString As String = "INSERT INTO
attribute_querypath(Concept_name, Attribute_name, Value)" & "
values(@Concept_name, @Attribute_name, @Value)"
Dim dbCommand As New Data.OleDb.OleDbCommand(commandString, dbConnection)
Dim conceptNameParam As New Data.OleDb.OleDbParameter
Dim attributeNameParam As New Data.OleDb.OleDbParameter
Dim valueParam As New Data.OleDb.OleDbParameter
conceptNameParam.ParameterName = "@Concept_name"
attributeNameParam.ParameterName = "@Attribute_name"
valueParam.ParameterName = "@Value"
conceptNameParam.Value = "Full教授"
attributeNameParam.Value = LblEmail.Text
dbCommand.ExecuteNonQuery()
dbConnection.Close()

Protected Sub TextBox4_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles txtboxResearchInterest.TextChanged
Dim connectionString As String = 
"provider=Microsoft.Jet.OLEDB.4.0;" & _
"data source=D:\university\university.mdb"
Dim dbConnection As New Data.OleDb.OleDbConnection
dbConnection.ConnectionString = connectionString
dbConnection.Open()

Dim commandString As String = "INSERT INTO
attribute_querypath(Concept_name, Attribute_name, Value)" & "
values(@Concept_name, @Attribute_name, @Value)"
Dim dbCommand As New Data.OleDb.OleDbCommand(commandString, dbConnection)
Dim conceptNameParam As New Data.OleDb.OleDbParameter
Dim attributeNameParam As New Data.OleDb.OleDbParameter
Dim valueParam As New Data.OleDb.OleDbParameter
conceptNameParam.ParameterName = "@Concept_name"
attributeNameParam.ParameterName = "@Attribute_name"
valueParam.ParameterName = "@Value"
conceptNameParam.Value = "Full教授"
attributeNameParam.Value = LblResearchInterest.Text
dbCommand.ExecuteNonQuery()
dbConnection.Close()
End Sub
Protected Sub CheckBox2_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles CheckBox2.CheckedChanged
    Dim connectionString As String = "provider=Microsoft.jet.oledb.4.0;" & _
        "data source=D:\university\university.mdb"
    Dim dbConnection As New Data.OleDb.OleDbConnection
    dbConnection.ConnectionString = connectionString
    dbConnection.Open()

    Dim commandString As String = "INSERT INTO attribute_querypath(Concept_name, Attribute_name, Query)" & _
        "values(@Concept_name, @Attribute_name, @Query)"
    Dim dbCommand As New Data.OleDb.OleDbCommand(commandString, dbConnection)
    Dim conceptNameParam As New Data.OleDb.OleDbParameter
    Dim attributeNameParam As New Data.OleDb.OleDbParameter
    Dim queryParam As New Data.OleDb.OleDbParameter
    conceptNameParam.ParameterName = "@Concept_name"
    attributeNameParam.ParameterName = "@Attribute_name"
    queryParam.ParameterName = "@Query"
    conceptNameParam.Value = "Full_professor"
    attributeNameParam.Value = LblName.Text
    queryParam.Value = "yes"
    dbCommand.Parameters.Add(conceptNameParam)
    dbCommand.Parameters.Add(attributeNameParam)
    dbCommand.Parameters.Add(queryParam)
    dbCommand.ExecuteNonQuery()
    dbConnection.Close()
End Sub

Protected Sub CheckBox3_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles CheckBox3.CheckedChanged
    Dim connectionString As String = "provider=Microsoft.jet.oledb.4.0;" & _
        "data source=D:\university\university.mdb"
    Dim dbConnection As New Data.OleDb.OleDbConnection
    dbConnection.ConnectionString = connectionString
    dbConnection.Open()

    Dim commandString As String = "INSERT INTO attribute_querypath(Concept_name, Attribute_name, Query)" & _
        "values(@Concept_name, @Attribute_name, @Query)"
    Dim dbCommand As New Data.OleDb.OleDbCommand(commandString, dbConnection)
    Dim conceptNameParam As New Data.OleDb.OleDbParameter
    Dim attributeNameParam As New Data.OleDb.OleDbParameter
    Dim queryParam As New Data.OleDb.OleDbParameter
    conceptNameParam.ParameterName = "@Concept_name"
    attributeNameParam.ParameterName = "@Attribute_name"
    queryParam.ParameterName = "@Query"
    conceptNameParam.Value = "Full_professor"
    attributeNameParam.Value = LblName.Text
    queryParam.Value = "yes"
    dbCommand.Parameters.Add(conceptNameParam)
    dbCommand.Parameters.Add(attributeNameParam)
    dbCommand.Parameters.Add(queryParam)
    dbCommand.ExecuteNonQuery()
    dbConnection.Close()
End Sub
Protected Sub LinkButton3_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnDepartment.Click
    Dim connectionString As String = "provider=Microsoft.jet.oledb.4.0;" & _
        "data source=D:\university\university.mdb"
    Dim dbConnection As New Data.OleDb.OleDbConnection
    dbConnection.ConnectionString = connectionString
    dbConnection.Open()

    Dim commandString As String = "INSERT INTO concept_querypath(Concept_name, Relationship_name)" & "
values(@Concept_name, @Relationship_name)"
    Dim dbCommand As New Data.OleDb.OleDbCommand(commandString, dbConnection)

    Dim conceptNameParam As New Data.OleDb.OleDbParameter
    conceptNameParam.ParameterName = "@Concept_name"
    conceptNameParam.Value = btnDepartment.Text
    conceptNameParam.DbType = Data.DbType.String
    dbCommand.Parameters.Add(conceptNameParam)

    Dim RelationshipNameParam As New Data.OleDb.OleDbParameter
    RelationshipNameParam.ParameterName = "@Relationship_name"
    RelationshipNameParam.Value = LblWork.Text
    RelationshipNameParam.DbType = Data.DbType.String
    dbCommand.Parameters.Add(RelationshipNameParam)

    dbCommand.ExecuteNonQuery()
    dbConnection.Close()
End Sub
Appendix E: Implementation of Ontology Mapping Algorithm

This appendix contains partially implementation code of the mapping algorithm. Appendix E.1 presents implementation of Jaro algorithm for comparison of two terms. Appendix E.2 presents main part of implementation of our ontology mapping algorithm. We use VB.NET for coding of algorithms.

Appendix E.1: The Coding of Mapping Function (Jaro algorithm)

Public Class Jaro

    Public Function Jaro(ByVal strA As String, ByVal strB As String) As Single
        Dim tokenA As String()
        Dim tokenB As String()
        Try
            tokenA = getTokens(strA)
            tokenB = getTokens(strB)
        Catch ex As Exception
            Return 0.0!
        End Try

        If IsNothing(tokenA) Or IsNothing(tokenB) Then
            Return 0.0
        End If

        If tokenA.Length <= tokenB.Length Then
            Jaro = match(tokenA, tokenB)
        Else
            Jaro = match(tokenB, tokenA)
        End If
    End Function

    ' split String into string array.
    Private Function getTokens(ByVal str As String) As String()
        Dim tokens As String() = Nothing
        ' if any other spacing character to support, just add here
        Dim sep(3) As Char
        sep(0) =заметка
        sep(1) =заметка
        sep(2) =заметка
        Try
            'Trace.WriteLine("String: " & str)
            If Not IsNothing(str) Then
                tokens = str.Split(sep, 4)
            End If
        Catch ex As Exception
            Catch ex As Exception
        End Try

End Class
Throw New Exception
End Try

Dim s As String
For Each s In tokens
    Trace.WriteLine("Concept:: " & s)
Next

TODO: jaro and jaro then take max jaro

getTokens = tokens
End Function

'caters for 1x1, 1x2, m x n where m <= n
'(as long as the 1st token length is less or equal with 2nd token)
'doesn't support 2x1, 3x2, m x n where m > n
Private Function match(ByVal a As String(), ByVal b As String()) As Single
    Dim aMax As Integer = 0
    Dim bMax As Integer = 0
    Dim isPerfect As Boolean = True
    Dim dataA As String
    Dim dataB As String
    Dim average(a.Length - 1, b.Length - 1) As Single
    Dim i As Integer = 0
    Dim j As Integer
    For Each dataA In a
        j = 0
        For Each dataB In b
            Dim longestLength As Integer = 0
            dataA = dataA.ToLower 'force letter to small letter
            dataB = dataB.ToLower
            If String.Compare(dataA, dataB) <> 0 Then
                isPerfect = False
                Exit For
            End If
            longestLength = longest(dataA, dataB)
            Dim matching(longestLength - 1) As Integer
            Dim transpose(longestLength - 1) As Integer
            Dim iLoop As Integer
            Dim jLoop As Integer
            'direct matching
            'Trace.WriteLine("comparing Token1: " & dataA)
            'Trace.WriteLine("comparing Token2:  " & dataB)
            For iLoop = 0 To dataA.Length - 1 Step 1
                'Trace.Write(iLoop)
                If iLoop <= dataB.Length - 1 Then
                    If dataA.Chars(iLoop) = dataB.Chars(iLoop) Then
                        matching(iLoop) = 1
                    End If
                End If
            Next
End If

Next

'loop base on a, and check if got matching, if not, then
go to loop b
'the loop b will only compare on none matching position
'transposing match

For iLoop = 0 To dataA.Length - 1 Step 1
    If matching(iLoop) <> 1 Then
        For jLoop = 0 To dataB.Length - 1 Step 1
            If matching(jLoop) <> 1 Then
                If dataA.Chars(iLoop) = dataB.Chars(jLoop) Then
                    transpose(jLoop) = 1
                    Exit For
                End If
            End If
        Next
    End If
Next

'begin______________________Add matching and
'transpose_______________________

Dim m As Integer
Dim matchValue As Integer = 0
Dim transposeValue As Integer = 0

'Trace.WriteLine("Token1: " & dataA)
'Trace.WriteLine("Token2: " & dataB)
'Trace.Write("Match:: ")
For m = 0 To matching.Length - 1 Step 1
    matchValue = matchValue + matching(m)
    'Trace.Write(matching(m))
Next

'Trace.WriteLine("")
'Trace.Write("Trans:: ")
For m = 0 To transpose.Length - 1 Step 1
    transposeValue = transposeValue + transpose(m)
    'Trace.Write(transpose(m))
Next

'Trace.WriteLine("")
'Trace.WriteLine("Match + Trans:: " & (matchValue + (transposeValue / 2)))

average(i, j) = (matchValue + (transposeValue / 2)) / longestLength

'Trace.WriteLine("Average:: " & (matchValue + (transposeValue / 2)) & "/" & longestLength & " = " & average(i, j))

'transpose_______________________Add matching and

j = j + 1
Next
i = i + 1
Next

Dim max As Single
Dim sumMax As Single = 0
If a.Length = 1 Then
    For i = 0 To a.Length - 1 Step 1
        max = 0
        Dim value As Single
        Dim k As Integer
        For k = 0 To j - 1 Step 1
            max = max + average(i, k)
        Next
        sumMax = sumMax + max
    Next
Else
    'from average(i,j) we have all the jaro for each compare.
    'take the max of each (i,x), than divide by dataA.length
    For i = 0 To b.Length - 1 Step 1
        max = 0
        For j = 0 To a.Length - 1 Step 1
            If average(j, i) > max Then
                max = average(j, i)
            End If
        Next
        sumMax = sumMax + max
    Next
End If
'Trace.WriteLine("Total Max: " & sumMax)
'Trace.WriteLine("Length to be devided::: " & b.Length)
match = sumMax / b.Length

End Function

Private Function longest(ByVal a As String, ByVal b As String) As Integer
    If a.Length > b.Length Then
        longest = a.Length
    Else
        longest = b.Length
    End If
End Function

End Class

Appendix E.2: The Coding of Ontology Mapping Algorithm
Dim MF As New Jaro
Dim mThreshold As New Single
Dim mSearchStr As String
Dim mMappedConcept As New ArrayList
Dim aMappingTableNew As New ArrayList
Dim aMappedAttribute As New ArrayList

Public Property mappedConcept() As ArrayList
    Get
        Return mMappedConcept
    End Get
    Set(ByVal value As ArrayList)
        mMappedConcept = value
    End Set
End Property

Public Property mappedAttribute() As ArrayList
    Get
        Return aMappedAttribute
    End Get
    Set(ByVal value As ArrayList)
        aMappedAttribute = value
    End Set
End Property

Public Function start() As DataTable
    Dim dataTable As New DataTable
    Dim ds As DataSet = mUniDao.getConceptQuery
    Dim dt As DataTable
    Dim dr As DataRow
    dt = ds.Tables(0)
    dr = dt.Rows(0) ' for first, second, third and forth, on applicable for the first record in query path
    mSearchStr = dr("Concept_name".ToString)
    'Trace.WriteLine("Concept Name:: " & mSearchStr)
    firstStep()
    secondStep()
    thirdStep()
    forthStep()

    Dim sim As New Similar()
    Dim simTemp As New Similar()

    For Each dr In dt.Rows
        ' if the number is 1, dont need to go for fifthstep and sixthstep
        'load concept into GenericConcept
        mSearchStr = dr("Concept_name".ToString)
        'Trace.WriteLine("is Sub concept:: " & mSearchStr)
        dr("is_subconcept".ToString)
        Dim queryConcept As GenericConcept = loadGeneric(mSearchStr)
        If Not (dr("Number").ToString = "1") Then
            'fifthStep() skip for now
            simTemp = sixthStep(sim, queryConcept, dr("is_subconcept".ToString))
Else
  'extract the first concept matching, will be used in step 6 above
  Dim cMappingArr As Similar() = CType(mSimilarAL.ToArray(GetType(Similar)), Similar())
  sim = cMappingArr(0)
End If
Next
startAttribute()
start = dataTable
End Function

Public Sub startAttribute()
  'Trace.WriteLine("StartAttribute processing...")
  'get all from attribute_query path. concept and the att
  'for each row in that,
  'find the matching concept, then match all uni att and synName with local concept's att name
  'if match, insert into the list
  'if no match, put 0
  Dim attributeAL As ArrayList = loadAttributeQuery()
  Dim queryAtt As UniAttributes
  For Each queryAtt In attributeAL
    Dim cMapping As CMappingTable
    For Each cMapping In Me.cMappingTableNew
      If queryAtt.name = cMapping.C1 Then    'concept name match
        Dim localAtt As LocalAttributes
        Dim localAttributes As ArrayList = loadLocalAttribute(cMapping.CiL)
        'check all attribute
        For Each localAtt In localAttributes
          Dim result As String
          Dim display As String = cMapping.C1 + " | " + queryAtt.attName + " --> "
          result = attMatch(queryAtt, localAtt)
          If Not result = "0" Then
            display = display + localAtt.name + " | " + result
            aMappedAttribute.Add(display)
          Else
            display = display + result 'the result is 0, meaning no match
            End If
        Next
      End If
    Next
  Next
End Sub

Public Function attMatch(ByVal queryAtt As UniAttributes, ByVal localAtt As LocalAttributes) As String
  Dim matchAttribute As String
  matchAttribute = "0" 'no match in the beginning
'if direct match
  Trace.WriteLine("Direct Att matching ::" & queryAtt.attName & " <U-L> " & localAtt.name)
  If MF.Jaro(queryAtt.attName, localAtt.name)  > mThreshold Then
    Trace.WriteLine("Direct Att matched :: " & queryAtt.attName & " <U-L> " & localAtt.name)
    matchAttribute = localAtt.name
    Return matchAttribute
  End If

'query match with all local's concept attname
Dim localAttName As String
For Each localAttName In localAtt.attName
  Trace.WriteLine("Query att VS all local att matching ::" & queryAtt.attName & " <U-L> " & localAttName)
  If MF.Jaro(queryAtt.attName, localAttName) > mThreshold Then
    Trace.WriteLine("Query att VS all local att matched ::" & queryAtt.attName & " <U-L> " & localAttName)
    matchAttribute = localAttName
    Return matchAttribute
  End If
Next

'local's attname match with all query's att syn name
Dim uniAttSynName As String
For Each uniAttSynName In queryAtt.synName
  Trace.WriteLine("Query att syn name VS local att matching ::" & uniAttSynName & " <U-L> " & localAtt.name)
  If MF.Jaro(uniAttSynName, localAtt.name) > mThreshold Then
    Trace.WriteLine("Query att syn name VS local att matched ::" & uniAttSynName & " <U-L> " & localAtt.name)
    matchAttribute = localAtt.name
    Return matchAttribute
  End If
Next

Dim uniSyn As String
Dim locAtt As String
For Each uniSyn In queryAtt.synName
  For Each locAtt In localAtt.attName
    Trace.WriteLine("Att Syn matching ::" & uniSyn & " <U-L> " & locAtt)
    If MF.Jaro(uniSyn, locAtt) > mThreshold Then
      Trace.WriteLine("Att Syn matched ::" & uniSyn & " <U-L> " & locAtt)
      matchAttribute = locAtt
      Return matchAttribute
    End If
  Next
Next

'loop all local
'  loop all uni
'  compare see got match. if got match, what do i do? return the matching att name

Return matchAttribute
Public Sub firstStep()
    'Trace.WriteLine("First Step processing....")
    Dim uniCon As New UniConcept
    Dim dr As DataRow

    Dim dt As DataTable
    ' call DaoUniversity to get the Concept with all it's synonyms name from the University DB
    Dim UniDs As DataSet = mUniDao.GetConcept(mSearchStr)
    dt = UniDs.Tables(0)

    For Each dr In dt.Rows
        Dim synNameAL As New ArrayList
        'synName(0) = dr("syn_name1").ToString 'syn could be null....ToString force it to empty string
        'synName(1) = dr("syn_name2").ToString
        'synName(2) = dr("syn_name3").ToString

        If Not String.IsNullOrEmpty(dr("syn_name1").ToString) Then
            synNameAL.Add(dr("syn_name1").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name2").ToString) Then
            synNameAL.Add(dr("syn_name2").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name3").ToString) Then
            synNameAL.Add(dr("syn_name3").ToString)
        End If

        Dim synName() As String = CType(synNameAL.ToArray(GetType(String)), String())
        uniCon.synName = synName
        uniCon.conceptId = dr("concept_id")
        uniCon.conceptName = dr("concept_name")

        '''Trace.WriteLine("Concept from DB: " & dr("concept_name"))
        '''Trace.WriteLine("syn_name1 from DB: " & dr("syn_name1"))
        '''Trace.WriteLine("syn_name2 from DB: " & dr("syn_name2"))
        '''Trace.WriteLine("syn_name3 from DB: " & dr("syn_name3"))
    Next
    'ds.Dispose() dont dispose now.. cause need to use on other step

    'get all ontology concepts from Local DB
    Dim localDs As DataSet = mLocalDao.GetAllConcept()
    dt = localDs.Tables(0)

    For Each dr In dt.Rows
        Dim localCon As New LocalConcept ' populate data
        Dim jaroResult As Single
        Dim attNameAL As New ArrayList
        'Dim attName(2) As String
        'attName(0) = dr("Attribute_name1").ToString
        'attName(1) = dr("Attribute_name2").ToString
        'attName(2) = dr("Attribute_name3").ToString
    Next
End Function
If Not String.IsNullOrEmpty(dr("Attribute_name1").ToString)
    Then
        attNameAL.Add(dr("Attribute_name1").ToString)
    End If
If Not String.IsNullOrEmpty(dr("Attribute_name2").ToString)
    Then
        attNameAL.Add(dr("Attribute_name2").ToString)
    End If
If Not String.IsNullOrEmpty(dr("Attribute_name3").ToString)
    Then
        attNameAL.Add(dr("Attribute_name3").ToString)
    End If

' 'Trace.WriteLine("length before convert: " & attNameAL.Count)
Dim attNameArr() As String = CType(attNameAL.ToArray(GetType(String)), String())
' 'Trace.WriteLine("length after convert: " & attNameArr.Length)

localCon.conceptId = dr("concept_id")
localCon.conceptName = dr("concept_name")
localCon.fatherName = dr("Father_name").ToString
localCon.attributeName = attNameArr

' jaro here...
' 'Trace.Write("Jaro:: " & dr("concept_name") & ", " & uniCon.conceptName & ", " & uniCon.conceptName & ", " & uniCon.conceptName & ", " & uniCon.conceptName & 
    jaroResult = MF.Jaro(localCon.conceptName, uniCon.conceptName)
' 'Trace.WriteLine("Jaro Result: " & jaroResult)
If jaroResult > mThreshold Then
    Dim same As New Similar
    same.jaroResult = jaroResult
    same.local = localCon
    same.university = uniCon
    'Trace.WriteLine("***Insert into Similarity AL***")
    'Trace.WriteLine("Local concept: " & localCon.conceptName)
    mSimilarAL.Add(same)
Else
    For i = 0 To uniCon.synName.Length - 1 Step 1
        ' 'Trace.Write(i)
        ' 'Trace.Write("Jaro Synname compare :: " & localCon.conceptName & ", " & uniCon.synName(i) & ", " & uniCon.synName(i) & 
            jaroResult = MF.Jaro(localCon.conceptName, uniCon.synName(i))
        ' 'Trace.WriteLine("Jaro Result SynName:: " & jaroResult)
        If jaroResult > mThreshold Then
            Dim same As New Similar
            same.jaroResult = jaroResult
            same.local = localCon
            same.university = uniCon
        End If
    Next i
End If
'Trace.WriteLine("**Insert into Similarity AL**")
' show syn name, then insert concept name (Ci and CiL)
mSimilarAL.Add(same)
End If
Next
End If
Next

'If String.IsNullOrEmpty(uniCon.synName(0).ToString) Then
'    'Trace.WriteLine("empty or null detected")
'End If
End Sub

'get father of conceptname and father of CiL from similarAL
Private Sub secondStep()
'Trace.WriteLine("Second Step processing....")
Dim father As String = mUniDao.getFather(mSearchStr)
Dim tempAL As New ArrayList
Dim similarArray() As Similar = CType(mSimilarAL.ToArray(GetType(Similar)), Similar())
Dim i As Integer
For i = 0 To similarArray.Length - 1 Step 1
    Dim localCon As LocalConcept = similarArray(i).local
    Dim result As Single = MF.Jaro(father, localCon.fatherName)
    similarArray(i).fatherMatching = result
    tempAL.Add(similarArray(i))
Next
Me.mSimilarAL = tempAL 'update latest array list with fathermatching info
End Sub

'third step.... compare attribute and relationship from Ci with attribute with CiL
Public Sub thirdStep()
'Trace.WriteLine("Third Step processing....")
Dim sim As Similar
' loop for every item inside CiL in SimilarAL
For Each sim In mSimilarAL
    'load the university's key property first (with either attribute or relationship)
    'Trace.WriteLine("ConceptName:: " & sim.university.conceptName)
    Dim ab As New UniAttributes
    Dim key As New KeyProperty
    sim.university.keyProperty = key
    sim.university.keyProperty.attribute = loadUniAttribute(sim.university.conceptName)
    sim.university.keyProperty.relationship = loadUniRelationship(sim.university.conceptName)
    Dim attributeStr As String
    Dim result As Single
    For Each attributeStr In sim.local.attributeName ' at least got 1 attribute name
        result = MF.Jaro(attributeStr, sim.university.keyProperty.attribute.name)
        If (result > mThreshold) Then
            'match found
        End If
    Next
Next
End Sub
sim.propertyMatching = 1
'Trace.WriteLine("detected! adding 1 attribute for 
& attributeStr)
Exit For
End If
result = MF.Jaro(attributeStr,
sim.university.keyProperty.relationship.name)
If (result > mThreshold) Then
  sim.propertyMatching = 1
  'Trace.WriteLine("detected! adding 1 relationship for 
" & attributeStr)
  Exit For
End If
Next
Next
For Each sim In mSimilarAL
  'Trace.WriteLine(sim.propertyMatching)
Next

'forth step, compare aggregation
Private Sub forthStep()
  'Trace.WriteLine("Forth Step processing....")
  'loop the msimilar, find max and put into cMappingTable
  Dim loopSim As Similar
  Dim maxSim As Similar

  For Each loopSim In mSimilarAL
    If IsNothing(maxSim) Then
      maxSim = loopSim
    End If
    If loopSim.propertyMatching + loopSim.fatherMatching > 
      maxSim.propertyMatching + maxSim.fatherMatching Then
      maxSim = loopSim
    End If
  Next
  'Trace.WriteLine("Best::" & maxSim.university.conceptName)
  mMappedConcept.Add(maxSim.local.conceptName)
  cMappingTAble.Add(maxSim)

  Dim map As New CMappingTable
  map.C1 = maxSim.university.conceptName
  map.CiL = maxSim.local.conceptName
  cMappingTableNew.Add(map)
End Sub

'firth step... if there is
Public Sub fifthStep()
  'Trace.WriteLine("Fifth Step processing....")
  Dim loopSim As Similar
  Dim attQueryAL As ArrayList = loadAttributeQuery()

  For Each loopSim In cMappingTAble
Dim attributeName_CiL As String

For Each attributeName_CiL In loopSim.local.attributeName
    'Trace.WriteLine("Attribute Name: " & attributeName_CiL)
    'take this att name, and match with the attribute from attribute query
    Dim attribute_C1 As UniAttributes
    For Each attribute_C1 In attQueryAL
        Dim match As New FifthMatch
        'Trace.WriteLine("Matching Uni att Name: " & attribute_C1.name & ", local att : " & attributeName_CiL)
        If MF.Jaro(attribute_C1.name, attributeName_CiL) > mThreshold Then
            match.C1 = attribute_C1.name
            match.CiL = attributeName_CiL
            mAttributeMappingTable.Add(match)
            'Trace.WriteLine("FifthStep Match found")
            'Trace.WriteLine("Uni att Name" & attribute_C1.name & ", local att : " & attributeName_CiL)
        Else
            Dim synName_C1 As String
            For Each synName_C1 In attribute_C1.synName
                'Trace.WriteLine("Matching Uni att SynName: " & synName_C1 & ", local att : " & attributeName_CiL)
                If MF.Jaro(synName_C1, attributeName_CiL) > mThreshold Then
                    match.C1 = attribute_C1.name
                    match.CiL = attributeName_CiL
                    mAttributeMappingTable.Add(match)
                    'Trace.WriteLine("FifthStep Match found")
                    'Trace.WriteLine("Uni att's syn Name" & synName_C1 & ", local att : " & attributeName_CiL)
                End If
            Next
        End If
    Next
Next

Private Function sixthStep(ByVal sim As Similar, ByVal qConcept As GenericConcept, ByVal isSubConcept As String) As Similar
    Dim newSimilar As New Similar
    'Trace.WriteLine("Sixth Step processing..")
    If isSubConcept = "False" Then
        Dim localDs As DataSet = mLocalDao.GetAllConcept()
        Dim dt As DataTable = localDs.Tables(0)
        Dim dr As DataRow
        For Each dr In dt.Rows
            If isUniMatch(qConcept, dr("concept_name")) Then
                'Trace.WriteLine("6.x Second Situation Match, local concept name:: " & dr("concept_name") & " Uni Concept Name :: " & qConcept.name)
                sim.local.conceptName = dr("concept_name")
            End If
        Next
    End If
End Sub
mMappedConcept.Add(dr("concept_name")) 'this is use to display on the Mapped Concept list

' this group here is to set for the attribute mapping later
Dim map As New CMappingTable
map.C1 = qConcept.name
map.CiL = dr("concept_name")
cMappingTableNew.Add(map)

Return sim
' compare all local concept name with name and synName of Uni concept
'match. return from here
End If
Next
End If

Dim isMatch As Boolean = False
Dim CiL As String = sim.local.conceptName

'Trace.WriteLine("Local Concept:: " & sim.local.conceptName)
Dim children As ArrayList = loadChildren(sim.local.conceptName)
Dim child As Children

For Each child In children
  'Trace.WriteLine("Children matching ::" & child.name & " <-> " & qConcept.name)
  If isUniMatch(qConcept, child.name) Then
    'Trace.WriteLine("6.1 Children match, local concept name:: " & child.name & " Uni Concept Name :: " & qConcept.name)

    sim.local.conceptName = child.name ' update the local concept name C2L (CnL), so the next C2 (Cn) will look for the correct C2L (CnL)
    mMappedConcept.Add(child.name) ' this is use to display on the Mapped Concept list
  End If
Next
End If

Else
Dim grandChildren As ArrayList = loadChildren(child.name)
Dim grandChild As Children

For Each grandChild In grandChildren
  'Trace.WriteLine("GrandChildren matching ::" & grandChild.name & " <-> " & qConcept.name)
  If isUniMatch(qConcept, grandChild.name) Then
    'Trace.WriteLine("6.1.1 Children match, local concept name:: " & child.name & " Uni Concept Name :: " & qConcept.name)

    sim.local.conceptName = child.name ' update the local concept name C2L (CnL), so the next C2 (Cn) will look for the correct C2L (CnL)
    mMappedConcept.Add(child.name) ' this is use to display on the Mapped Concept list
  End If
Next
End If
Else
Dim map As New CMappingTable
map.C1 = qConcept.name
map.CiL = child.name
cMappingTableNew.Add(map)

sixthStep = sim ' return from here **
Return sim

Else
Dim grandChildren As ArrayList = loadChildren(child.name)
Dim grandChild As Children

For Each grandChild In grandChildren
  'Trace.WriteLine("GrandChildren matching ::" & grandChild.name & " <-> " & qConcept.name)
  If isUniMatch(qConcept, grandChild.name) Then
    'Trace.WriteLine("6.1.1 Children match, local concept name:: " & child.name & " Uni Concept Name :: " & qConcept.name)

    sim.local.conceptName = child.name ' update the local concept name C2L (CnL), so the next C2 (Cn) will look for the correct C2L (CnL)
    mMappedConcept.Add(child.name) ' this is use to display on the Mapped Concept list
  End If
Next
End If
Else
Dim map As New CMappingTable
map.C1 = qConcept.name
map.CiL = grandChild.name
cMappingTableNew.Add(map)

sim.local.conceptName = grandChild.name ' update the local concept name C2L (CnL), so the next C2 (Cn) will look for the correct C2L (CnL)
mMappedConcept.Add(grandChild.name) 'this is use to display on the Mapped Concept list
sixthStep = sim 'return from here **
Return sim
End If
Next
End If

Dim siblingAL As ArrayList = loadSibling(sim.local.conceptName)
Dim sibling As Sibling
For Each sibling In siblingAL
  'Trace.WriteLine("sibling matching ::" & sibling.name & " <-> " & qConcept.name)
  If isUniMatch(qConcept, sibling.name) Then
    'Trace.WriteLine("6.2 Sibling match, local concept name:: " & sibling.name & " Uni Concept Name :: " & qConcept.name)
    Dim map As New CMappingTable
    map.C1 = qConcept.name
    map.CiL = sibling.name
    cMappingTableNew.Add(map)

    sim.local.conceptName = sibling.name
    mMappedConcept.Add(sibling.name)
    isMatch = True
    Exit For
  End If
Next

If isMatch Then
  sixthStep = sim
End If

' 'Trace.WriteLine("Direct matching ::" & sim.local.conceptName & " <-> " & qConcept.name)
If isUniMatch(qConcept, sim.local.conceptName) Then
  'Trace.WriteLine("6.3 Direct concept match, local content name:: " & sim.local.conceptName & " Uni Concept Name :: " & qConcept.name)
  Dim map As New CMappingTable
  map.C1 = qConcept.name
  map.CiL = sim.local.conceptName
  cMappingTableNew.Add(map)

  sim.local.conceptName = sim.local.conceptName
  mMappedConcept.Add(sim.local.conceptName)
  isMatch = True
Else
  'Trace.WriteLine("6.4 match (meaning no one match), uni content name:: " & qConcept.name)
Dim map As New CMappingTable
map.C1 = qConcept.name
map.CiL = "0"
cMappingTableNew.Add(map)

'sim.local.conceptName = "null" 'dont set to null here...
instead, just show on UI
'also because the next match will look for the previous local
concept match
mMappedConcept.Add("0")
End If

sixthStep = sim
End Function

Public Sub New(ByVal mLocalDao As DaoLocal, ByVal mUniDao As
DaoUniversity)
    MyBase.new()
    Me.mLocalDao = mLocalDao
    Me.mUniDao = mUniDao
    Me.mThreshold = 0.6!
End Sub

Private Function loadUniAttribute(ByVal str As String) As UniAttributes
    Dim attribute As New UniAttributes
    Dim ds As DataSet = mUniDao.getAttribute(str)
    Dim dt As DataTable
    Dim dr As DataRow
    dt = ds.Tables(0)
    For Each dr In dt.Rows
        Dim synNameAL As New ArrayList
        If Not String.IsNullOrEmpty(dr("syn_name1").ToString) Then
            synNameAL.Add(dr("syn_name1").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name2").ToString) Then
            synNameAL.Add(dr("syn_name2").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name3").ToString) Then
            synNameAL.Add(dr("syn_name3").ToString)
        End If
        Dim synName() As String =
            CType(synNameAL.ToArray(GetType(String)), String())
        attribute.conceptId = dr("concept_id")
        attribute.synName = synName
        attribute.name = dr("name")
        ''Trace.WriteLine("attribute from DB: " & dr("name"))
        ''Trace.WriteLine("Syn Length: " &
        attribute.synName.Length)
        ''Trace.WriteLine("syn_name1 from DB: " & dr("syn_name1"))
        ''Trace.WriteLine("syn_name2 from DB: " & dr("syn_name2"))
        ''Trace.WriteLine("syn_name3 from DB: " & dr("syn_name3"))
        Next
    loadUniAttribute = attribute
End Function

Private Function loadUniRelationship(ByVal str As String) As UniRelationship
    Dim relationship As New UniRelationship
    Dim ds As DataSet = mUniDao.getRelationship(str)
    Dim dt As DataTable
    Dim dr As DataRow
    dt = ds.Tables(0)
    For Each dr In dt.Rows
        Dim synNameAL As New ArrayList
        If Not String.IsNullOrEmpty(dr("syn_name1").ToString) Then
            synNameAL.Add(dr("syn_name1").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name2").ToString) Then
            synNameAL.Add(dr("syn_name2").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name3").ToString) Then
            synNameAL.Add(dr("syn_name3").ToString)
        End If
        Dim synName() As String = CType(synNameAL.ToArray(GetType(String)), String())
        relationship.conceptId = dr("concept_id")
        relationship.synName = synName
        relationship.name = dr("name")
        'Trace.WriteLine("attribute from DB: " & dr("name"))
        'Trace.WriteLine("Syn Length: " & relationship.synName.Length)
        'Trace.WriteLine("syn_name1 from DB: " & dr("syn_name1"))
        'Trace.WriteLine("syn_name2 from DB: " & dr("syn_name2"))
        'Trace.WriteLine("syn_name3 from DB: " & dr("syn_name3"))
    Next
    loadUniRelationship = relationship
End Function

Private Function loadAttributeQuery() As ArrayList
    Dim attQueryAL As New ArrayList
    Dim ds As DataSet = mUniDao.getAttributeQuery()
    Dim dt As DataTable
    Dim dr As DataRow
    dt = ds.Tables(0)
    For Each dr In dt.Rows
        Dim attribute As New UniAttributes
        Dim synNameAL As New ArrayList
        If Not String.IsNullOrEmpty(dr("syn_name1").ToString) Then
            synNameAL.Add(dr("syn_name1").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name2").ToString) Then
            synNameAL.Add(dr("syn_name2").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name3").ToString) Then
            synNameAL.Add(dr("syn_name3").ToString)
        End If
        Dim synName() As String = CType(synNameAL.ToArray(GetType(String)), String())
        attribute.attributeId = dr("attribute_id")
        attribute.synName = synName
        attribute.name = dr("name")
        'Trace.WriteLine("attribute from DB: " & dr("name"))
        'Trace.WriteLine("Syn Length: " & attribute.synName.Length)
        'Trace.WriteLine("syn_name1 from DB: " & dr("syn_name1"))
        'Trace.WriteLine("syn_name2 from DB: " & dr("syn_name2"))
        'Trace.WriteLine("syn_name3 from DB: " & dr("syn_name3"))
    Next
    loadAttributeQuery = attQueryAL
End Function
End If
Dim synName() As String = CType(synNameAL.ToArray(GetType(String)), String())

attribute.synName = synName
attribute.name = dr("concept_name")
attribute.attName = dr("Attribute_name")
attQueryAL.Add(attribute)

"Trace.WriteLine("attribute from DB: " & dr("concept_name"))
"Trace.WriteLine("Syn Length: " & attribute.synName.Length)
"Trace.WriteLine("syn_name1 from DB: " & dr("syn_name1"))
"Trace.WriteLine("syn_name2 from DB: " & dr("syn_name2"))
"Trace.WriteLine("syn_name3 from DB: " & dr("syn_name3"))
Next
loadAttributeQuery = attQueryAL

End Function

Private Function loadLocalAttribute(ByVal id As String) As ArrayList
Dim attQueryAL As New ArrayList

Dim ds As DataSet = mLocalDao.getAttribute(id)
Dim dt As DataTable
Dim dr As DataRow
dt = ds.Tables(0)
For Each dr In dt.Rows
Dim attribute As New LocalAttributes
Dim attNameAL As New ArrayList

If Not String.IsNullOrEmpty(dr("Attribute_name1").ToString) Then
attNameAL.Add(dr("Attribute_name1").ToString)
End If
If Not String.IsNullOrEmpty(dr("Attribute_name2").ToString) Then
attNameAL.Add(dr("Attribute_name2").ToString)
End If
If Not String.IsNullOrEmpty(dr("Attribute_name3").ToString) Then
attNameAL.Add(dr("Attribute_name3").ToString)
End If

attName() As String = CType(attNameAL.ToArray(GetType(String)), String())

attribute.attName = attName
attribute.name = dr("concept_name")
attQueryAL.Add(attribute)

"Trace.WriteLine("attribute from DB: " & dr("concept_name"))
"Trace.WriteLine("Att Length: " & attribute.attName.Length)
"Trace.WriteLine("att_name1 from DB: " & dr("Attribute_name1"))
"Trace.WriteLine("att_name2 from DB: " & dr("Attribute_name2"))

loadAttributeQuery = attQueryAL
''Trace.WriteLine("att_name3 from DB: " &
  dr("Attribute_name3"))
Next
loadLocalAttribute = attQueryAL
End Function

Private Function loadChildren(ByVal id As String) As ArrayList
  Dim attQueryAL As New ArrayList
  'Trace.WriteLine("loadChildren ID: " & id)
  Dim ds As DataSet = mLocalDao.GetChildren(id)
  Dim dt As DataTable
  Dim dr As DataRow
  dt = ds.Tables(0)
  For Each dr In dt.Rows
    Dim children As New Children
    Dim synNameAL As New ArrayList
    children.name = dr("Concept_name")
    attQueryAL.Add(children)
  Next
loadChildren = attQueryAL
End Function

Private Function loadSibling(ByVal id As String) As ArrayList
  Dim attQueryAL As New ArrayList
  'Trace.WriteLine("loadSibling ID: " & id)
  Dim ds As DataSet = mLocalDao.GetSibling(id)
  Dim dt As DataTable
  Dim dr As DataRow
  dt = ds.Tables(0)
  For Each dr In dt.Rows
    Dim sibling As New Sibling
    Dim synNameAL As New ArrayList
    sibling.name = dr("Concept_name")
    attQueryAL.Add(sibling)
  Next
loadSibling = attQueryAL
End Function

Private Function isUniMatch(ByVal concept As GenericConcept, ByVal
conceptName As String) As Boolean
  Dim isMatch = False
  If MF.Jaro(concept.name, conceptName) > mThreshold Then
    Return True
  Else
    'get the synName from Uni
    Dim synName_C1 As String
    For Each synName_C1 In concept.synName
      'Trace.WriteLine("SynName matching ::" & conceptName & " <L-U> " & synName_C1)
If MF.Jaro(synName_C1, conceptName) > mThreshold Then
    Return True
End If
Next
End If

isUniMatch = isMatch
End Function

Private Function loadGeneric(ByVal name As String) As GenericConcept
    Dim ds As DataSet = mUniDao.GetConcept(name)
    Dim dt As DataTable
    Dim dr As DataRow
    dt = ds.Tables(0)
    Dim concept As New GenericConcept
    For Each dr In dt.Rows
        Dim synNameAL As New ArrayList
        If Not String.IsNullOrEmpty(dr("syn_name1").ToString) Then
            synNameAL.Add(dr("syn_name1").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name2").ToString) Then
            synNameAL.Add(dr("syn_name2").ToString)
        End If
        If Not String.IsNullOrEmpty(dr("syn_name3").ToString) Then
            synNameAL.Add(dr("syn_name3").ToString)
        End If
        Dim synName() As String = CType(synNameAL.ToArray(GetType(String)), String())
        concept.name = dr("concept_name")
        concept.synName = synName
        Exit For
    Next
dloadGeneric = concept
End Function

End Class
Appendix F: Runtime Complexity of Algorithm

In this appendix we explain the calculation of the complexity levels which occur when doing mapping of query path terms to terms of local ontology. We consider \( n \) as the number of elements (entities) in a local ontology (ontology size). The runtime of a single entity or fixed set and length of entities is independent of the size of the ontology and we consider complexity of \( O(1) \). Any step which requires access to the whole path of an ontology graph (acyclic rooted graph) leads to a complexity of \( O(\log n) \) (depth of a path) and any step needs access to the whole ontology results in the complexity of \( O(n) \).

We calculate the complexity of each step of the algorithm in the worst case as follows:

**Complexity of MF (Mapping Function) used in algorithm:** We used Jaro-Winkler algorithm (Winkler, 1999) for Mapping Function (MF). The runtime complexity of MF depends on the string length of two terms and is \( O(L_1 \cdot L_2) \) where \( L_1 \) and \( L_2 \) are length of two terms. In our algorithm we consider a maximum fixed length for terms. Therefore the complexity of MF is \( O(MF) = O(1) \);

**Complexity of First Step, Name Matching:** Root of query path \( C_1(name) \) and all its synonyms names \( C_1(syn-name) \) are compared with all the local ontology concepts \( C_L(name) \).

\[
\text{for all } C_L(name) <> \text{null do } \quad O(n)
\]
\[
\begin{align*}
\text{If } \text{MF}[C_1(name),C_iL(name)] &\geq \text{threshold then } \quad O(1) \\
\text{similarity-table} &\leftarrow (C_i, C_{iL}); \quad O(1)
\end{align*}
\]
\[
\text{Else: for all } C_1(syn-name) <> \text{null do } \quad O(\text{fix-number}) = O(1)
\]
\[
\begin{align*}
\text{If } \text{MF}[C_1(syn-name),C_L(name)] &\geq \text{ threshold then } \quad O(1) \\
\text{Add } (C_i, C_{iL}) \text{ to similarity-table; exit; } \}
\end{align*}
\]

**Total:** \( O(n) \cdot O(1) = O(n) \);

**Complexity of Second Step, Father Matching:** in this step father and grandfather of \( C_1 \) are compared with father and grandfather of each \( C_iL \) (\( C_{iL} \) in similar table). In the worst case
we may have n candidate mapping terms (all local ontology terms).

If (father of C1 and CiL <> null) then: O(n)

Father-matching(C1,CiL) \leftarrow MF[father(C1),father(CiL)]; O(1)

If (grandfather of C1 and CiL <> null) then: O(n)

{Father-matching(C1,CiL) += [MF(grandfather(C1),grandfather(CiL))];
Father-matching(C1,CiL) /= 2; } O(1)

Therefore the complexity for comparison of the father is O(n).O(1) = O(n) and for grandfather is O(n).O(1) = O(n). Total complexity for this step of algorithm is O(n) + O(n) = O(n);

**Complexity of Third Step, Key-Property Matching:** In this step key properties of concepts are compared with all attributes and relationships of candidate mapping terms.

while C_{iL} (name)<>null do O(n)

{ for all A,R of C_{iL} do O(fix-number) = O(1)
If each MF[C1(key-property-name & key-property-syn-names), C_{iL}(A & R)] >= threshold then C_{iL}(similar-property) +1;} O(1)

In the implementation of the algorithm we consider a fixed number of key property for each concept element of the domain ontology. The number of attributes and relationships for each concept element in the local ontology is independent of the ontology size. For the implementation of this step we consider a maximum fixed number of attributes and relationships for each concept element in order to reduce the runtime complexity of the algorithm. Thence the complexity in the worst case is: O(n).O(1).O(1) = O(n);

**Complexity of Fourth Step, Aggregation:** In this step the algorithm finds the most similar corresponding concept with C1 among all C_{iL} from the similar table. For this purpose, the algorithm aggregates the results of the father matching (second step) and key-property matching (third step) for each C_{iL}. In the worst case the number of candidate mapping terms is “n”.

212
For all CiL do 

\[ O(n) \]

\{ 
Weight(CiL) \leftarrow father-matching(CiL) + CiL(similar-property); \}

CiL \leftarrow CiL[ \text{MAX} (CiL-weight) > \beta] \quad O(1)

C-mapping-table \leftarrow (C_i, CiL); \quad O(1)

Thence the complexity in the worst case is: \( O(n).O(1) = O(n); \)

**Complexity of Fifth Step, Attribute Matching:** This step compares attributes of a concept with all attributes and relationships of the mapping term. In the worst case we can consider \( m \) attributes and relationships so the complexity of this step is \( O(m); \)

\[
\text{while } C_i(A-name_i)<>\text{null do } \quad O(m) \\
\{ \text{If } MF(C_i(A-name_i), CiL(A-name or R-name)) \geq \text{threshold } \text{the Add } \}
\{ \text{(C_i(A-name), CiL(A-name or R-name)) to att-mapping-table; } \quad O(1) \}
\text{Else: while } A-syn-name_i<>\text{null do } \quad O(\text{fix}) = O(1) \\
\{ \text{If } MF(C_i(A-syn-name_i), CiL(A-name or R-name)) \geq \text{threshold } \text{then } \}
\text{Add } (C_i(A-name), CiL(A-name or R-name)) \text{ to att-mapping-table; Else } i \leftarrow i+1; \} \} \quad O(1)
\]

**Complexity of Sixth Step, Next Concept Matching:** In the worst situation of this step we need to execute step 1, 2, 3 and 4 between next concept of the query path and all concepts of the local ontology. Therefore complexity of next concept mapping is the total complexity of steps 1, 2, 3 and 4 as follows:

\[ O(n) + O(n) + O(n) + O(n) = O(n); \]

The sixth step of algorithm is repeated in order to find mapping terms for all other next query path terms. The complexity of this iteration depends on the length of the query path. The domain ontology uses acyclic directed rooted graph which we can consider “\( \log n \)” for query path size. Therefore the total complexity of the sixth step is \( O(n \cdot \log n); \)

**Complexity of the Seventh Step, Query Concept Matching:** In the worst situation of this step we need to execute steps 1, 2, 3 and 4 between the query concept and all concepts of the local ontology. Therefore the complexity of this step is:

\[ O(n) + O(n) + O(n) + O(n) = O(n); \]
Complexity of step 8, query rewriting: In this step, the algorithm rewrites the user query with discovered corresponding terms from the local ontology. This step has complexity of $O(1)$.

Aggregation of algorithm complexity regards to mentioned above complexities is:

$$O(n) + O(n) + O(n) + O(m) + O(n \cdot \log n) + O(n) + O(1) = O(n \cdot \log n)$$
Appendix G: Publications

This thesis is based on papers presented at conferences published during the research as listed below:


References

(Antonion and Harmelen, 2003)

(Arens et al., 1997)

(Bayardo et al., 1997)
Bayardo, R. J., Bohrer, W., Brice, R., Cichochi, A., Fowler, J., Kashyap, V. and et al. (1997). InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments, ACM SIGMOD.

(Beneventano et al., 2004)

(Berners-Lee, 2001)

(Berners-Lee, 1998)

(Bishr, 1997)

(Bishr et al., 1999)

(Bornhovd and Buchmann, 2000)

(Bray et al., 2008)
(Brickley and Guha, 2004)

(Chalupsky, 2000)

(Chawathe et al., 1994)

(Chen et al., 2000)

(Choi et al., 2005)

(Cohen et al., 2003)

(Dean et al., 2003)

(Decker et al., 1999)

(Ding et al., 2002)

(Doan, et al., 2003)
(Dumbill, 2001)

(Ehrig and Staab, 2004)

(30 Farquhar et al., 1996)

(Ehrig and Sure, 2004)

(Farquhar et al., 1996)

(Fensel, 2001a)

(Fensel, 2001b)

(Fensel et al., 2001)

(Fensel et al., 1999)

(Fileto, 2001)

(Garcia-molina et al., 1997)
(Garcia-Solaco et al., 1996)

(Genesereth et al., 1997)

(Geppert and Hakimpour, 2001)

(Goh, 1997)

(Goh et al., 1999)

(Gruber, 1995)

(Gruber, 1993)

(Guarino, 1998)

(Gustavsson, 2001)

(Hakimpour, 2003)
(Huhns and Singh, 1997)

(Ibrahim and Schwinger, 2001)

(Kashyap and Sheth, 1998)

(Kim and Seo, 1991)

(Kim et al., 1993)

(Klein, 2001)

(Klyne and Carroll, 2004)

(Lopez, 1999)

(Madhavan et al., 2002)

(Maedche, 2002)
(McCann et al., 2003)

(McGuinness and Harmelen, 2003)

(McGuinness, et al., 2000)

(Meersman, 1994)

(Melnik et al., 2002)

(Mena et al., 2000)

(Miller, 1995)

(Mitra and Wiederhold, 2002)

(Mosses, 1990)

(MySQL, 2003)
(Noy, 2004)

(Noy et al., 2000)

(Noy and Musen, 2003)

(Noy and Musen, 2002)

(Noy and Musen, 2001)

(Noy and Musen, 2000)

(Noy and McGuinness, 2001)

(OGC, 1999)

(Palmer, 2001)

(Park and Ram, 2004)
(Patel-Schneider et al., 2004)

(Pouyan, 2003)

(Quass et al., 1995)

(Rahm and Bernstein, 2001)

(Rosenthal and Sciore, 1995)

(Silva and Rocha, 2003)

(Sheth, 1998)

(Smith et al., 2003)

(Sowa, 2002)

(Stuckenschmidt et al., 2000)
(Thanh-Le et al., 2004)

(Uschold, 1996)

(Uschold, and King, 1995)

(Visser et al., 2000)

(Visser, 1997)

(Wache et al., 2000)

(Wache et al., 1999)

(Wang and Kim, 2002)

(Widom, 1995)
(Winkler, 1999)

(Woods, 1975)

(W3schools Tutorial, 2008a)

(W3schools Tutorial, 2008b)

(W3schools Tutorial, 2008c)

(Xiaomeng, 2004)