# QUERYING SEMANTICALLY HETEROGENEOUS DATA SOURCES USING ONTOLOGIES

by

ADITI BREED

M.S., UNIVERSITY OF MUMBAI, 2005

A THESIS

submitted in partial fulfillment of the requirements for the degree

**MASTER OF SCIENCE**

**Department of Computing and Information Sciences**
**College of Engineering**

**KANSAS STATE UNIVERSITY**
**Manhattan, Kansas**

2008

Approved by:

Major Professor
Dr Doina Caragea

# Abstract

In recent years, we have witnessed a significant increase in the number, size and diversity of the available data sources in many application domains. Data sources in a particular domain are autonomously created and maintained, and therefore distributed and semantically heterogeneous. In this thesis, we focused on the problem of querying such semantically heterogeneous data sources from a user's perspective. We approach this problem by using the concepts of ontologies and mappings between ontologies. A system for answering queries in a transparent way to the user has been designed and implemented. The main components of this system are an ontology mapping algorithm that maps user ontologies to data source ontologies, and a query processing engine that maps user queries to queries that can be answered by the data sources in the system. We have shown that machine learning algorithms can also be incorporated in the system, thus making it possible to learn machine learning classifiers (in particular, generative models such as Naïve Bayes) from distributed, semantically heterogeneous data sources. Because many data sources today are relational in nature, in this work we have dealt specifically with relational data sources, as opposed to flat files, XML or object oriented data sources. However, our system can be easily extended to other types of data sources.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgements

## Dedicated to My Parents

This thesis is the outcome of many contributions in addition to significant individual efforts and I wish to thank everyone for their help and guidance.

I am thoroughly indebted to my advisor, *Dr. Doina Caragea*, for being a constant source of guidance, support and advice at all times. I would like to thank her for introducing me to my research topic and for keeping my interest alive throughout the ups and downs of the research phase. Her guidance has been fundamental to my academic progress and of paramount importance to the successful conclusion of this challenging topic. Over the past two years, she has always made me feel at home and has endured many questions at various times. She has also invested considerable time and effort in reviewing this document, helping to enhance its clarity and conciseness.

I also wish to thank *Dr. Hankley* and *Dr. Amtoft* for accepting to serve on my thesis committee. In addition to contributing to my personal and academic development, they have always showered their kindness upon me, which has made studying at KSU an enjoyable experience.

I acknowledge the Computer and Information Sciences Department at KSU for giving me an opportunity to pursue my Masters degree and for providing me the necessary resources that helped materialize nascent ideas into an accomplished thesis. I would also like to thank my colleagues in the Computer Science Department, in particular *Vinodkumar Mudaliar, Pranshu Gupta* and *Claudia Chaudhari*, all of whom have endured my questions and opinions throughout these two years. Also a kind vote of thanks to my professors in the Physics Department at the D.G. Ruparel College (Mumbai, India), in particular *Dr. Tushar Desai, Dr. Ajit Naik* and *Mrs. Vidya Patil* who have been a constant source of encouragement and support. I would also like to thank *Jagruti Chedda* who originally introduced me to the concept of DBMS, in particular, Oracle.

And last but definitely not the least, I would like to thank my family and in particular my brother, *Dr. Aniket Breed* without whom I would have fallen astoundingly short of pursuing something so ambitious.

# CHAPTER 1 - Introduction

## 1.1 **Background**

The traditional view of information systems as data intensive applications has changed due to rapid and continuous advances in technology. Thus, in addition to the ability to deal with large amounts of data, current information systems are heterogeneous in their representation and scope, types of data that they can handle interaction with other systems or with users, etc.

Principled use of multiple information systems together, to achieve a global task, calls for system integration at several levels. In particular, it calls for the integration of data from distributed, autonomous and heterogeneous data sources [11]. Such data sources exist in many application domains, including biological sciences, space sciences, environmental sciences, advertisement, ecommerce and banking, manufacturing, where information systems are required to support flexible querying and retrieval, among other tasks.

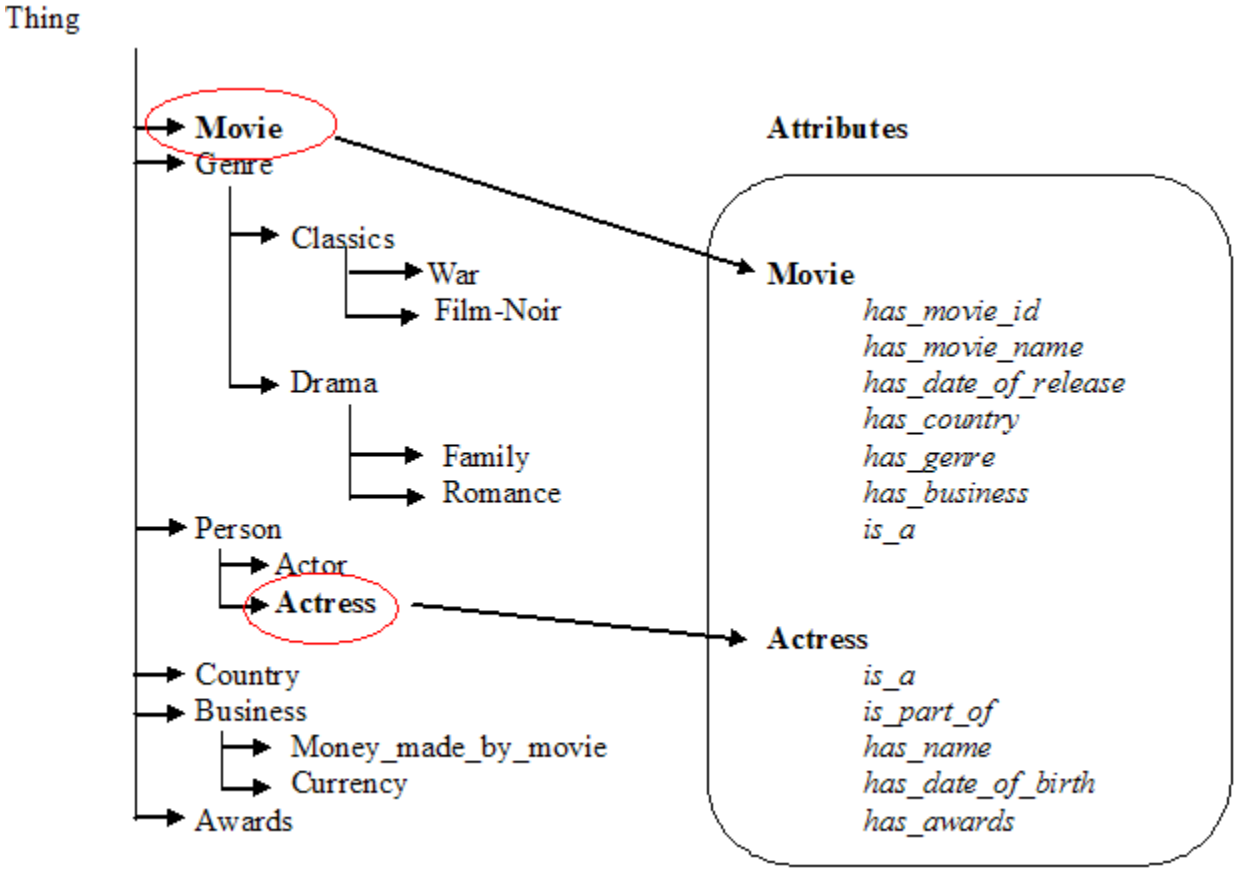
At a high level, our goal is to design and implement a system for querying semantically heterogeneous relational databases, where each autonomous database uses its own terminology to describe its data. We will approach this task by assuming that there are metadata, in the form of *ontologies*, associated with each database. These metadata make the database and its content self-describing. In addition, *mappings* between user and database ontologies will be used to answer queries from semantically heterogeneous databases from a user's perspective.

The work described in this thesis relies on emerging semantic web concepts, in particular *ontologies* and *mappings* between ontologies, and their representation languages. This is because information integration challenges like those addressed in this work, have been influential in the birth of the semantic web, where data is assumed to have structure and semantics, and ontologies are used to describe the semantics of the data. An *ontology* is a description (like a formal specification of a program) of the concepts and relationships among concepts that exist in a particular domain [2]. They can be seen as a way to organize data. Such organization could make it possible for software agents to "understand" the semantics of these data and intelligently locate and integrate them for a wide variety of tasks. *Sharing common understanding of the structure of*

*information among people or software agents* is one of the more common goals in developing ontologies [2].

The complexity of an ontology varies with the expressiveness of the concepts and relationships it can represent. In this work, we will use simple *ontologies* that can be seen as *concept hierarchies*, with each *concept* having a set of (possibly hierarchical) *attributes* that describe it. The *relationships* between concepts are either *part-of* or *is-a* relationships. **Figure 1.1** shows an example of a fragment of a concept hierarchy in the *movie-marketing* domain (this domain will be used as an example and as an application throughout this thesis).

**Figure 1.1: Concept hierarchy in the movie-marketing domain.**




In addition to their role in describing information sources, ontologies play a key role in enabling interoperability among distributed information systems. However, this requires appropriately reconciling multiple ontologies. The reconciliation is done through the means of *mappings* between ontologies. Mappings can be found manually or semi-automatically [**7**], by

identifying mismatches between ontologies at the *lexical* level (different terminologies, e.g. *production company* versus *studio*), at the *structural* level (different levels of abstraction, e.g. *company* versus *production company* or *music company*) or at the *data representation* level (e.g., *currency* can be represented in *American dollars* or *Euros*).

## 1.2 Problem Definition and Contributions

We will introduce the problem that we address in this work through an example from the movie-marketing domain. Assume that there are several databases that store related but not completely overlapping information about movies (e.g., International Movie Database - IMDB, Yahoo! Movies, MSN Movies, Foreign Films). These databases are autonomously created and maintained, which means that each has its specific ontology. A movie marketing company, having yet its own ontology, wants to analyze these databases together to find out what makes a movie successful. More precisely, they want to build a predictive model and use it to predict if a new movie will be successful or not. As shown in [12], the task of building predictive models, such as naïve Bayes or decision tree classifiers can be reduced to the task of answering statistical queries, e.g. COUNT queries, from data. In the movie-marketing scenario, a naïve Bayes classifier can be constructed by answering a set of COUNT queries from semantically heterogeneous data sources. Given the data source ontologies and the user ontology (in our case, the marketing company ontology), together with mappings from the user ontology to the data source ontologies, count queries can be answered in a transparent way to the user, as if the data were all at a central location and described in the user ontology.

Therefore, in this thesis, we design and implement a system that can answer queries from semantically heterogeneous movie data sources. We use answers to such queries to build naïve Bayes classifiers for predicting if a movie is successful or not. More precisely, the whole process involves the following steps:

1. Creating and populating two movie ORACLE databases.
2. Creating ontologies (i.e., concept hierarchies) that describe the two movie databases and also a user ontology.
3. Designing an algorithm for finding mappings between user ontology and data source ontologies.

4. Designing an engine for answering SQL queries from semantically heterogeneous data sources, through the means of ontologies and mappings between ontologies.
5. Evaluating the ability of the system to answer complex SQL queries from semantically heterogeneous data (especially, in situations where data in different sources is specified at different levels of abstraction).
6. Evaluating the ability of the system to build naïve Bayes classifiers in a way transparent to the user.

# CHAPTER 2 - Querying Semantically Heterogeneous Data Sources

The primary goal of this thesis is to build a system that can be used to answer queries from autonomous data sources. In this chapter, we will describe the main concepts and ideas beyond the query answering engine that we build.

## 2.1 Ontology Extended Databases and Mappings

In our framework, we assume that there are several ontology-extended databases,$(D_1,O_{D1}),\ldots, (D_n,O_{Dn})$, registered with an information system, and users (e.g., movie critiques or movie advertisement companies), $(U_1,O_{U1}),\ldots,(U_k,O_{Uk})$ that want to query the data available in the system, according to their own ontologies. **Figure 2.1** shows simple examples of two such databases and **Figure 2.2** shows fragments of their underlying movie ontologies, as well as a user movie ontology.

**Figure 2.1: Movies data collected by two movie critics in two different databases**

Table : Movies

| | Movie ID | Movie Title | Date of Release | Country | Alternate Name |
|---|---|---|---|---|---|
| D1 | M01 | ABC | 01-01-08 | USA | Abc,The |
| | M02 | DEF | 12-12-07 | Russia | Def,The |

Table : Genre

| MovieID | Genre_name |
|---|---|
| M10 | Action |
| M25 | Comedy |

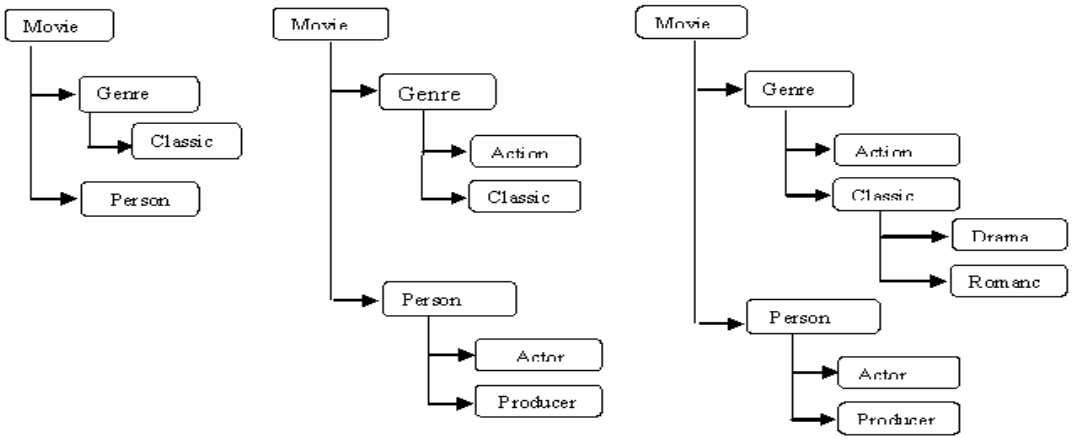
Table : Movie

| | MID | Movie Name | Release Date | Country Of Origin | Alternate Title |
|---|---|---|---|---|---|
| D2 | M01 | ABC | 01-01-08 | USA | Abc,The |
| | M02 | DEF | 12-12-07 | Russia | Def,The |

Table : genre

| MID | genre |
|---|---|
| M10 | Horror |
| M25 | Animation |

**Figure 2.2: User ontology $O_U$ and data source ontologies $O_{D1}$, $O_{D2}$associated with the movie concept.**

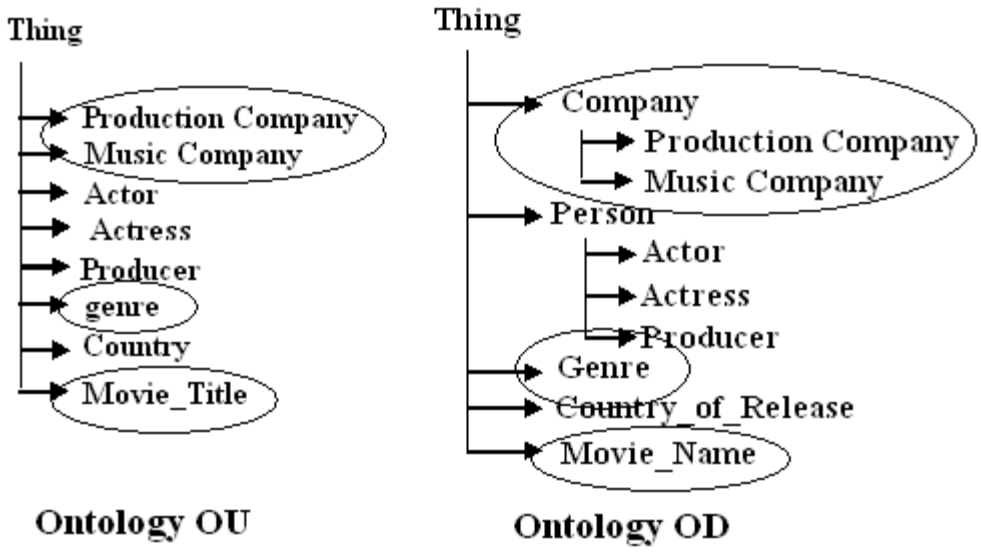



We assume that for each user $U_i$ and each ontology $D_j$, there is a mapping $\Phi_{ij}$ from the user ontology $O_{Ui}$ (source) to the data source ontology $O_{Dj}$(target). This mapping specifies how terms in $O_{Ui}$ are mapped to terms in $O_{Dj}$. Furthermore, the mapping can also specify conversion functions (e.g., to convert between different units). We will use the example in **Figure 2.3** to define the types of mismatches that we will try to capture in this work (although other types of mismatches can be defined). In particular, we consider mismatches that can be identified at several levels, as described below:

- Lexical mismatches (naming conflicts): when the same concept has different names in different ontologies. For example, in **Figure 2.3** the movie ontology $O_U$ identifies a movie by its *title*, while the ontology $O_D$ identifies a movie by its *name*. Naming conflicts are extremely difficult to resolve. The user or a software agent needs to have prior information or background knowledge about the concept to resolve such a conflict [13].

- Structural mismatches: when data in different databases are specified at different levels of abstraction. For example in **Figure 2.3**, *production company* is at a lower level of abstraction in $O_D$ than it is in the ontology $O_U$(more specific than *company*).
- Representation mismatches: when mismatches occur due to differences in the representation (e.g., different units) of the data. In this case, mappings can be seen as conversion functions.

**Figure 2.3: Types of mismatches between terms in user ontology and terms in database ontology**




A simple algorithm for finding lexical and structural mismatches has been designed as part of this thesis. The ideas behind the algorithm will be briefly described in the next section. We also deal with semantic heterogeneity at the representation level in the system we build, but we use standard conversion functions (e.g., currency conversions) for this type of mappings.

## 2.2 Finding Mappings Between Ontologies

The types of semantic mappings considered in this thesis can be written as follows [14]*:*

(1)$O_U$:$x$ = $O_D$:$y$($x$ in $O_U$ is semantically equivalent to $y$ in $O_D$),

(2) $O_U$:$x \leq O_D$:$y$(x is semantically subsumed by y), and

(3) $O_U$:$x \geq O_D$:$y$($x$ semantically subsumes y).

**Figure 2.4** shows examples of such mappings between a user ontology $O_U$ and the databases ontologies $O_{D1}$ and $O_{D2}$, respectively.

**Figure 2.4: Structural and lexical mismatches between terms in the user ontology $O_U$ and terms in two databases $D_1$ and $D_2$.**

| **$O_U \rightarrow O_{D1}$** | **$O_U \rightarrow O_{D2}$** |
|---|---|
| ID: $O_U$ →MovieID: $O_{D1}$ | ID: $O_U$ → MID: $O_{D2}$ |
| Name: $O_U$ → Movie Name: $O_{D1}$ | Name: $O_U$ → Movie Title: $O_{D1}$ |
| Name: $O_U$ → Alternate Name: $O_{D1}$ | |
| DOR: $O_U$ → Date: $O_{D1}$ | DOB: $O_U$ → Date: $O_{D2}$ |
| Genre: $O_U$ → Genre: $O_{D1}$ | Genre: $O_U$ → Genre Type: $O_{D2}$ |
| Country: $O_U$ → Country of Origin: $O_{D1}$ | Country: $O_U$ → Country: $O_{D2}$ |
| Language: $O_U$ → Language: $O_{D1}$ | Language: $O_U$ → Language of Movie: $O_{D2}$ |

Mapping type (1) means that the concept x in ontology $O_U$ is equivalent to the concept y in ontology $O_D$. For the purpose of our algorithm, we define the equivalence of concepts based on the equivalence of the attributes associated with these concepts.

Mapping type (2) means that that the concept x in ontology $O_U$ is a subtype of the concept y in ontology $O_D$. We define the subtype relation between concepts as the subset relation between their attributes. Thus, if the set of attributes of concept x is a subset of the set of attributes of concept y, then we say that x is a subtype of y.

Mapping type (3) means that that the concept x in ontology $O_U$ is a super-type of the concept y in ontology $O_D$. The super-type relation between two concepts is defined as the superset relation between their corresponding attributes.

The rules (1), (2), (3) can also be applied to find mappings between attributes associated with concepts, where the equivalence, subtype and super-type relations are defined in terms of the values of the attributes.

### *2.2.1 Resolving Structural Heterogeneity*

The structural heterogeneity in ontologies can be resolved by considering the types of components and structures residing in the ontology graphs [12]. Methods that use such information rely on the intuition that elements of two distinct entities (concept/attributes) are similar when other elements adjacent to them are also similar [15]. We will set the following criteria, adopted from [12], to decide if two entities A and B in an ontology are structurally similar:

1. The direct super-entities of A and B are similar.
2. The direct sub-entities of A and B are similar.
3. All entities on the paths from the root to the entities A and B, respectively, are similar.
4. A and B have similar attributes (for concepts) or similar values (for attributes).

### *2.2.2 Resolving Lexical Heterogeneity*

In semantic matching of ontologies, a linguistic analysis of the names of the entities and the concepts is done, using WordNet [16].Thus, a name-based check will always be performed before measuring the similarity of a pair of concepts by comparing the attributes which uniquely define the concepts.

More details on how we implement the ideas above are provided in the chapter on experimental design.

## 2.3 Querying Semantically Heterogeneous Databases

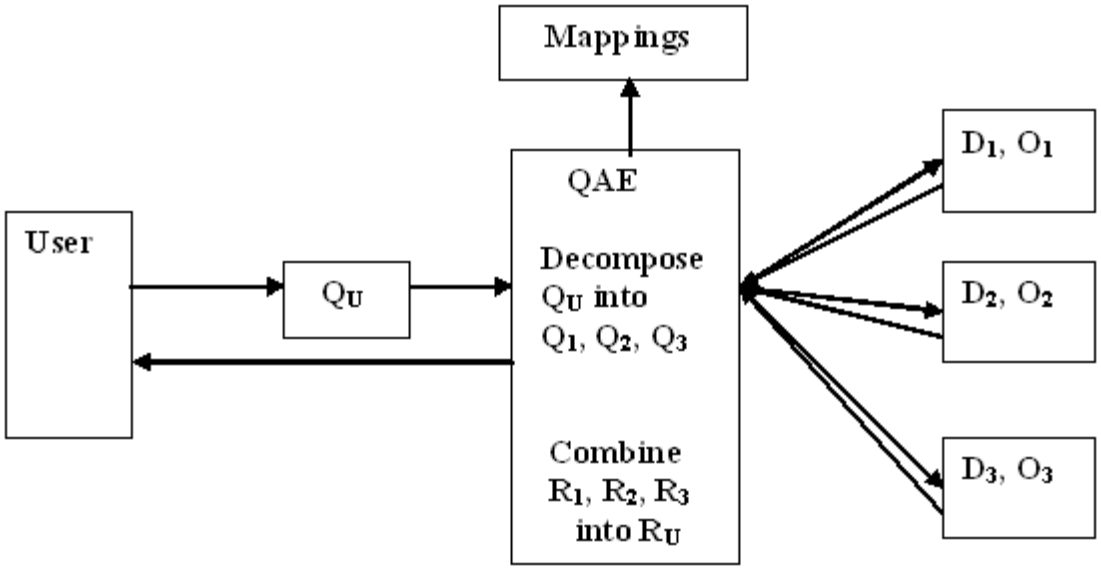
We should note that the mappings $\Phi_{ij}$ from user ontologies $O_{Ui}$ to data source ontologies $O_{Dj}$, enable a user to see a set of semantically heterogeneous databases as a single centralized homogeneous database that can be queried using the user ontology.

When a user registers with the system his/her ontology needs to be specified. Once it is specified, the system will use the ontology mapping procedure to generate candidate mappings between the user ontology and the ontologies of the databases available to the system. The resulting mappings are shown to the user and can be edited, if changes need to be made. Once

the user approves the mappings, they are stored in a mapping repository that is accessible by the query answering engine.

**Figure 2.5** shows the process the system follows in order to answer queries posed by a particular user U. Thus, a user query $Q_U$ is sent to the query answering engine (QAE), which has access to the mappings between the user ontology and the ontologies of the databases available in the system. The QAE decomposes the user query $Q_U$ in sub-queries, $Q_1,\ldots,Q_n$, that can be answered by the individual databases, $D_1,\ldots, D_n$, respectively. As the initial user query $Q_U$ is expressed in the user ontology $O_U$, sub-queries need to be translated to the database ontologies before they are sent to databases for execution. This is also done by the QAE by making use of the repository mappings corresponding to the user $O_U$. Finally, the results $R_1,\ldots,R_n$ that are received back from the distributed databases are combined by the QAE into a final result set $R_U$, which is returned to the user.

**Figure 2.5: Query answering process**




## 2.4 Partially Specified Data

It may be useful to explore what it means to answer a statistical query in a setting in which autonomous data sources differ in terms of the levels of abstraction at which data are described. We illustrate some of the issues that have to be addressed using an example:

Consider the data source ontologies $O_1$ and $O_2$ and the user ontology $O_U$ shown in **Figure 2.2**. The attribute *has_genre* in data source $D_2$ is specified in greater detail (lower level of abstraction) than the corresponding attribute *has_Genre_name* is in $D_1$. The data source $D_2$ carries information about the precise type of genre of Movies e.g., *Action, Classic* (which has subtypes *Romance and Drama*). However data source $D_1$ makes no distinctions between the types of genre *Drama* and *Romance,* i.e. it only has the class *Classics*. Suppose that $D_2$ contains 10 and 30 instances of movies with genre *Romance* and *Drama,* respectively, which are categorized are subclasses of the genre *Classic*. Suppose $D_1$ contains 80 instances of movies with genre *Classic*. However, the Ontology $O_1$ does not make an explicit mention of the subclasses *Romance* and *Drama*. Assume that the following statistical query $q_1$is posed by the user against the two data sources $D_1$ and $D_2$ based on user ontology $O_U$: What fraction of movies have genre *Classic*? The answer to this query can be computed in a straightforward fashion for $D_2$ as the number of movies with genre *Drama* and *Romance* (resulting in 10 + 30 instances) and for $D_1$ as the number of movies with genre *Classics* (which is 80 instances).

Let us consider a different statistical query $q_2$: What fraction of the movies in the two data sources have genre *Drama*? The answer to this query is not as straightforward as the answer to the previous query $q_1$. This is due to the fact that the genre *Drama* of movies in data source $D_1$ is only *partially specified* [14] [15] with respect to the ontology $O_U$. Consequently, we can never know the precise fraction of movies that have genre *Drama* based on the information available in the two data sources. But we assume that the proportion of movies with the genre *Drama* among the movies with genre *Classic* is similar in the two data sources. Thus, we assume that the data in data source $D_1$ is modeled by the same underlying distribution as the data source $D_2$.We can then *estimate* the fraction of *movies with genre Drama* in the data source $D_2$ based on the fraction of *movies with genre Drama* in the data source $D_1$ (i.e., 10 out of 40 i.e. 0.25) and use the result to answer the query $q_2$. Hence, the *Movies with genre Drama* admits in $D_1$ is 10 i.e. 25% of the movies with genre Classic. Thus, the answer to the query $q_2$ is (80 * (25/100)) = 20.

It should be noted that query $q_1$ is completely determined whereas the answer to the query $q_2$ is only partially determined by the data source ontologies $O_1$, $O_2$, the user ontology $O_U$ shown in **Figure 2.2** and the mappings in **Figure 2.3.** In such cases, answering statistical queries from semantically heterogeneous data sources requires the user to supply not only the mapping between the ontology and the ontologies associated with the data sources but also *additional*

*assumptions of a statistical nature* (e.g., that movies with same genre in $D_1$ and $D_2$ can be modeled by the same underlying distribution). The validity of the answer returned depends on the validity of the assumptions and the soundness of the procedure that computes the answer based on the supplied assumptions.

## 2.5 Learning Naïve Bayes Classifiers from Semantically Heterogeneous Databases

To demonstrate the usefulness of our system in learning predictive models from semantically heterogeneous distributed data sources, we show how a Naïve Bayes classifier can be linked to the system and how it can benefit from the ability of the system to answer queries from such data sources.

Because the data sources in our framework are relational, we need to consider a relational machine learning algorithm in our work. In particular, we demonstrate the feasibility of the proposed approach to building predictive models from semantically heterogeneous databases using relational variants of the classical naïve Bayes algorithm. This algorithm and its variants use Bayes theorem [19][20] to find the class that has the maximum a posterior probability (MAP) given a new instance, whose label needs to be predicted. We use the approach in [21] to transform the naïve Bayes algorithm into a relational learning algorithm and explore two different methods for estimating the posterior probabilities:

**Independent Value** (INDVAL): In this case, the data in the relational tables is arranged in such a way that each attribute in a record may have multiple values. Each value in the set of multi- attribute value set in the table is assumed to be independent of each other. An instance of each value in the multiset of attribute values if created to decide the most probable class using the following relation:

$$c_{MAP}(x) = \underset{c_j \in C}{\operatorname{argmax}} p(c_j) \prod_i \prod_{v_k \in V_i} p(v_k|c_j).$$

**Average Probability** (AVGPROB): This case is similar to the INDVAL case, in that it assumes that, each value in the set of multi –of attribute values if created to decide the most probable class using the following relation:

$$c_{MAP}(x) = \underset{c_j \in C}{\operatorname{argmax}}\, p(c_j) \prod_i \frac{\sum_{v_k \in V_i} p(v_k|c_j)}{|V_i|}.$$

As can be seen from these formulas, the calculation of the posterior probabilities does not require access to the raw data. Instead only counts of the attributes (attribute=value $v_k$& class=value $c_j$) are needed. These counts can be obtained using our query answering system in a transparent way to the user.

Thus, predictive models can be learned from distributed, heterogeneous databases using the query answering engine that we implement in our system.

# CHAPTER 3 - Related Work

This thesis has two important parts: finding mappings between ontologies (in particular, between a user ontology and data source ontologies) and querying semantically heterogeneous data sources through the means of mappings between ontologies (by mapping a user query to queries that databases can "understand").

In this chapter, we will review work related to the two main parts of the thesis. Thus, we will describe several systems for querying multiple heterogeneous distributed data sources (which may or may not use ontologies). We will also discuss various ontology mapping techniques to give the reader an idea about the variety of the methods currently available.

## 3.1 Systems for Querying Multiple, Distributed, Heterogeneous Data Sources

### 3.1.1 Querying Heterogeneous Data Sources Using Query Correspondence Assertions

The work by Ulf Leser and his colleagues [22] address the problem of querying heterogeneous data sources using query correspondence assertions. He defines a global schema to represent the knowledge in the heterogeneous data sources and a language to express the correspondences between the various data source and the global schema [22].He makes use of the concept of mediator schema, which subsumes the necessary parts of the source schemas. The mediator is aided by an interface, which wraps the heterogeneity in the heterogeneous sources, making available a *source schema* for each heterogeneous source and an asset of queries against the schema. The mediator will then answer queries by combining the data from the heterogeneous data sources available via the source schema. Thus, the correspondences between the mediator schema and the global schema can be used to formulate global queries. This approach does not use ontologies to represent the data sources, but instead uses schemas as a formal representation of data. The advantage of this approach is the use of query correspondence

assertions, which are more expressive than the content descriptions, thereby allowing the user to formulate complex queries against the data sources.

### 3.1.2 Query Processing for Heterogeneous Data Integration using Ontologies

Huiyong Xiao [23] suggests the use of ontologies as formal representations for XML and RDF heterogeneous data sources. He proposes to resolve the problem of syntactic, semantic and schematic heterogeneity in ontologies by integrating data in the data sources, using multiple 'local' ontologies. A single global ontology provides a conceptual view over all schematically heterogeneous data source. A thesaurus, formalized in terms of ontology, can be used for the mapping process to facilitate its automation. The interface provides support for high–level queries. However in this case, the heterogeneous data sources are only XML based sources, as opposed to databases, which are used in the real world for large and distributed datasets. This approach is very similar to our approach to query heterogeneous data source, except for the use of XML based data sources.

### 3.1.3 OBSERVER

The OBSERVER framework [24] designed by Eduardo Mena, Arantza Illarramendi provides answers to user defined queries using an incremental approach in a Global Information System. They suggest an Inter-ontology Relationships Manager, as a solution to the vocabulary sharing problem. This is a repository for relationships among the various terms in the multiple ontologies. The data, which captures the semantic metadata of the information in the ontologies, is organized as a lattice. The framework also includes an ontology server, which provides the term definitions in the ontology and helps retrieve data underlying the ontology for the query processor to further process the query. The framework also allows the user to browse through a set of ontologies and decide which ontology, the user wants to use as the "user" ontology. This is similar to our system, where the user can view the ontology and query the data sources.

### 3.1.4 OntoShare

OntoShare [25] is an ontology-based WWW knowledge-sharing environment for a community of users that models the interests of each user in the form of a user profile. OntoShare assigns a set of topics and ontological concepts depending on the user's interest and accordingly assigns a user profile. It has the capability to extract information from the World Wide Web and sources, which may have been made available by the user. This information is further shared with other users having similar interests depending on their profiles.

OntoShare uses ontologies to store the shared information, with a shared document leading to the creation of a new ontology represented in RDFS or RDF. RDFS is used to specify the classes in the ontology and their properties [25]. RDF is then used to populate this ontology with instances as information is shared [25]. However, it may be noted that it is impossible to render the content of a document exhaustively by an RDF description.

Thus, OntoShare, while not claiming to actually capture tacit knowledge, provides an environment which actively encourages the sharing of tacit knowledge [25].

### 3.1.5 Information Manifold

Levy, Rajaraman, Ordille [26] have designed the Information Manifold, a system that provides uniform access to heterogeneous data sources to answer complex queries. It tackles the problem of querying a large number of different data sources by providing a mechanism to describe declaratively the contents and query capabilities of available information sources. Since every data source has a different mode of interaction, the algorithm proposed in this work, uses the source descriptions to prune efficiently the set of information sources for a given query and to generate executable query plans. The most important feature of their proposed architecture and algorithm is that they scale up well to several hundred information sources.

### 3.1.6 SEWASIE

Dongilli, Fillottrani, Franconi, Tessaris [27] have proposed the design of the SEWASIE (Semantic Webs and AgentS in Integrated Economies) project. The project aims at using a multi agent system to query heterogeneous data sources with ontologies. The system proposes to use an advanced search engine enabling intelligent access to heterogeneous data sources on the web, in a rich semantic (ontological) framework. The system concentrates on the process of query

answering by designing a query tool that assists users in building queries and moving through different types of agents that collaborate in collecting the information for the answer.

The SEWASIE network implements a virtual network whose nodes are SEWASIE information nodes or better known as SiNodes [27].

The main characteristics of SEWASIE include: a multi layer data integration system, query building assisted by a tool, query rewriting using local ontologies. The architecture also includes monitoring tools integrated in the architecture. The most important feature of SEWASIE is that the SiNodes apply not only to unstructured, semi-structured data but also to relational databases, which is also a feature of our system architecture.

### 3.1.7 MOMIS

MOMIS stands for “Mediator environment for Multiple Information Sources" which was developed by the database research group at the University of Modena by Reggio Emilia [28][29][30][31].

It is a mediator-based system which can be used with structured and semi-structured data sources, to extract and integrate information. It approaches the problem of information integration on the basis of metadata of the information sources.

The MOMIS system relies on local and global schema of the data sources, which is unlike what we have in our system, where ontologies provide formal representation for the data sources. The system semi-automatically generates a set of mapping correspondences between the concepts in the local schema and the global schema.

MOMIS adds details to its mapping correspondences by calculating the “affinity coefficient” between concepts and groups similar classes into clusters using hierarchical clustering algorithms.

## 3.2 Ontology Mapping

Our query processing engine, as other query engines, depends crucially on the mappings established between the user ontology and the multiple heterogeneous data sources. Thus, the mapping technique is a significant part of the system architecture.

In this section, we will introduce several selected ontology mapping systems and will describe their functionality on the basis of their approach to resolve semantic and syntactic heterogeneity. We proceed with some motivating applications where ontology mapping plays an important role.

The concept of mapping has a range of meanings, including integration, unification, merging, mapping, etc. Mapping is defined a set of formulae that provide the semantic relationships between the concepts in the models [32].

However, Noy & Musen [33] believe that a mapping establishes correspondences among ontologies, and determines 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 [24], it is stated that the aim of mapping is to map concepts in the various ontologies to each other, so that a concept in ontology corresponds to a query (i.e., view) over the other ontologies.

We define ontology mapping as determining the set of correspondences between the user ontology and the multiple data source ontologies, by resolving the semantic and structural heterogeneity, in the context of this thesis. The Ontology mapping process can be broadly divided into two stages. The first stage involves the discovery of correspondences between ontology elements, while the second stage involves defining the discovered mappings so that other components can make use of them.

The mapping correspondences are produced in roughly two ways:

1. Applying a set of matching rules or
2. Evaluating similarity measures that compare a set of possible correspondence and help to choose valid correspondence from them. These heuristics often use syntactic information such as the names of the concepts or nesting relationships between concepts. They also use semantic information such as the inter-relationship between concepts (slots of frames in [34]), the types of the concepts, or the labeled-graph structure of the models [35].

The process of ontology mapping cannot be fully automated. However, automated tools can provide plausible mapping correspondences between ontologies using graph based inference rules or text mining, speeding up the process significantly.

In what follows we will describe a number of systems that can be used to support manual and semi-automatic ontology mapping. Due to the close relation of ontology mapping and merging, systems used for merging are also used for mapping ontologies. Thus, we provide a brief survey of various ontology mapping and merging tools.

### 3.2.1 Automatic Ontology Mapping

F. Wiesman et al. [36] proposed a system for ontology mapping to facilitate effective agent communication [37]. The paper suggests that lack of standardization hampers communication between agents. According to the authors, communication between these agents can be regularized by resolving the semantic and structural heterogeneity in the various information sources.

The authors introduce the concept of language games for mapping ontologies. The concept involves agents determine *joint attention* by finding an instance of a concept known by both agents communicating with each other. This system makes good use of AI reasoning for ontology mapping purposes and has also achieved some good results in setup of an experiment [26].

### 3.2.2 GLUE A Machine Learning Based Ontology Mapping System

GLUE [38] employs machine learning techniques to semi-automatically create semantic mappings between two ontologies. Probabilistic techniques are used to find similar concepts in multiple ontologies. GLUE uses domain constraints and background knowledge of the concepts to assist in the mapping process. The key feature of GLUE is that it uses multiple learning strategies, each of which exploits a different type of information either in the data instances or in the taxonomic structure of the ontologies [38].

The architecture of GLUE consists of three main modules: Distribution Estimator, Similarity Estimator, and Relaxation Labeler. The Distribution Estimator takes as input two ontologies and applies machine learning techniques to compute the joint probability distribution for every pair of concepts. Thus, instead of estimating specific similarity values directly, GLUE focuses on computing the joint distributions. The Distribution Estimator uses a variety of base learners and a meta-learner, incorporating a multi strategy learner approach. Thus, GLUE is able to achieve higher classification accuracy and hence better approximations of the joint distributions. The implementation of GLUE has two case learners, namely Content Learner and

Name Learner, and a meta-learner that is a linear combination of the base learners. Thus the output from this module is a similarity matrix between the concepts in the two taxonomies [38].

The Relaxation labeler takes as input the similarity matrix with the domain specific constraints and heuristic knowledge to search for mapping correspondences which best satisfy the observed similarities. The technique of Relaxation labeling is used to solve the problem of assigning labels to nodes of a graph, given a set of constraints [38].

The approach used by GLUE, unlike any other mapping tool, uses a single heuristic approach to map ontologies and multiple machine learning approaches to map ontologies. However the system is slow in its mapping process, which requires further research to locate the proper balance between accuracy and speed [39].

### 3.2.3 QOM - Quick Ontology Mapping

QOM optimizes the Naïve Ontology Mapping approach, which emphasizes the efficiency of mapping ontologies as opposed to the accuracy of the mapping correspondences generated by the mapping technique. Thus, QOM takes into account not only the quality but also the speed and efficiency of the mapping operation. The mapping algorithm is dependent on the number of candidate mapping pairs in the multiple ontologies, using a dynamic programming approach, which uses ontological structures to reduce the number of candidate mappings.

The features of this system have been summarized in [39] as follows,

- Optimizing the mapping operation for efficiency decreases overall mapping quality
- Labels are the most important features for mapping.
- Combining many feature matching approaches leads to significantly higher quality mappings
- QOM shows very good results and quality is lowered only marginally
- QOM is faster than other approaches by a factor of 10 to 100 times

### 3.2.4 Ontology Mapping using Background Knowledge

Zharko and Klein in [40] have proposed a system which finds mapping correspondences between concepts, using structure rich ontologies as background knowledge. This approach is useful in scenarios where ontologies are assumed to be a list of concepts and have no particular

structure. This approach for ontology mapping may be considered as innovative and first of its nature that exploits background or domain knowledge for the mapping purpose.

In this thesis, we use WordNet to map concepts in the ontologies, assuming that the labels of the properties and the concepts represent the meaning of the "concepts" in natural language.

### 3.2.5 Natural Language Processing Techniques for Ontology Mapping

D Fossati et al. [41] approach the problem of ontology mapping from a computational linguistic point of view and have presented a natural language processing (NLP) based mechanism for ontology mapping. This technique of mapping is particularly useful to resolve the problem of instance heterogeneity.

### 3.2.6 Chimaera

Chimaera [23] is a web based ontology merging and diagnosis tool developed by the Stanford University Knowledge Systems Laboratory (KSL). It is particularly targeted at smaller ontologies, by merging ontologies produced by multiple authors.

Chimera supports two major tasks in merging ontologies [24]

1. Coalesce two semantically identical terms from different ontologies, such that they are referred to by the same name in the resulting ontology, and
2. Identify terms that are related by subsumption, disjointness, or instance relationships and provide support for introducing those relationships.

Chimera allows the user to map ontologies by suggesting terms which are possible candidates in the ontologies to be merged or have taxonomic relationships which are to be included in the merged ontology.

### 3.2.7 PROMPT: Automated Ontology Merging and Alignment Tool

Noy, N.F. and Musen, M.A [42] developed a tool that provides a semiautomatic

approach to merging and aligning ontologies. The tool is also available as a plug-in in Protégé. This tool guides the user to remove inconsistencies in ontologies by determining the conflicts in the ontologies and suggests solutions.

The PROMPT algorithm guides the user in creating an integrated ontology, by creating an initial list of matches based on class names. The algorithm then waits for the users response on the suggestions, and makes further changes based on the type of the operation. Next, the algorithm creates a list of mapping suggestions, which are based on the structure of the ontologies. These are further verified by the user for the tools to implement the changes.

Thus, the tool is interactive and does not create an integrated ontology by applying mapping rules directly, but instead asks the user for permission to implement the suggestions. Also it uses an incremental approach to provide a complete list of mapping suggestion, using the knowledge to learn the concepts in the ontologies at each level. Noy, N.F. and Musen, M.A [42] have presented the following as the fundamental features of the PROMPT tool:

- Setting the preferred ontology.
- Maintaining user's focus.
- Providing feedback to the user.
- Logging and reapplying the operations

In this chapter, we have analyzed tools for querying multiple heterogeneous data sources and compared them with our tool. It has been observed that although some tools used ontologies to counter the problem of heterogeneity among the data sources, each tool has a differential approach to using these ontologies for resolving this heterogeneity. Since ontology mapping is crucial in our system, we have also evaluated presently available mapping techniques, thereby motivating the use of ontologies to resolve heterogeneity. It has been observed that the current ontology mapping techniques, algorithms and merging and alignment applications use diverse ideas from the field of machine learning, knowledge engineering to find semantic correspondences among the candidate ontologies. These ontology techniques exploit the semantics, structure and syntax of the ontologies.

Our evaluation of query and ontology mapping tools has lead us to believe that it is possible to design simple heuristic methods for finding mapping correspondences between

multiple ontologies. In the next chapter, we will systematically describe the system that we have developed to query multiple heterogeneous data sources, including the mapping technique which is a crucial part of the system.

# CHAPTER 4 - System Design and Architecture

In this chapter, we describe the design and architecture of our system for querying multiple heterogeneous data sources. We state the assumptions made with respect to the development and functionality of the system. We describe the resources that we exploit in this system and explain why they were chosen over other possible candidates. As the ontology mapping procedure is one of the main contributions of this thesis, we also discuss it in detail in this chapter. At last, we describe all system components in detail and introduce the user to the functionality of the system.

## 4.1 Ontology Representation Language

We will make the following assumptions about the ontologies that will be used in the system:

1. They represent overlapping knowledge in the movie-marketing domain.
2. They are represented using the Web Ontology Language (OWL), sublanguage OWL Lite (described below).

The **Web Ontology Language** is a language for defining and instantiating *Web ontologies* [43].It is the language recommended by the World Wide Web Consortium. An *OWL ontology* will include descriptions of *classes*, *properties* (or *attributes*) and their *instances*.

The OWL language provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users:

- *OWL Lite* is intended to provide classification hierarchy and simple constraint features. For example, *OWL Lite* supports cardinality constraints, but permits only cardinality values of 0 or 1. *OWL Lite* is particularly useful for resolving structural semantic heterogeneity between hierarchy-like ontologies.
- *OWL DL* supports maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems [43]. *OWL DL* is so named due to its correspondence with *description logics* [43].

- *OWL Full* supports maximum expressiveness and the syntactic freedom of RDF, but does not provide any computational guarantees. This language is difficult to support using most reasoning software. All valid RDF documents are *OWL Full*.

We use the *OWL Lite* sublanguage for defining the ontologies in our system, because *OWL Lite* provides the structural features and the expressivity necessary to define hierarchical ontologies and mappings between them. Given that we use hierarchical, tree-like ontologies, we will assume that the OWL ontologies are in the form of XML trees. This assumption is particularly useful finding mapping rules semi-automatically. Please note that concepts are called *classes* in OWL and attributes are called *properties,* so we will use these names when talking about concepts and attributes in the OWL context. Furthermore, we will refer to concept/classes/attributes/properties as *nodes* in the graph hierarchy they belong to, when talking about them in the graph context.

## 4.2 Ontology Mapping and Structural Heterogeneity

Our goal is to map two hierarchical ontologies $O_U$ and $O_{D1}$ that describe the user U and data sources $D_1$, respectively. These ontologies may not contain the same number of entities (concept/attributes). Furthermore, their corresponding hierarchies may not have similar structures. However, they describe the same domain and have the same extensions. Thus, in terms of number of concepts that ontologies $O_U$ and $O_D$ contain, there are three possible cases and we handle all these cases in our work:

a) Ontology $O_U$ has same number of concepts as ontology $O_D$.
b) Ontology $O_U$ has fewer concepts as compared to the number of concepts in ontology $O_{D.}$
c) Ontology $O_U$ has more concepts as compared to the number of concepts in ontology $O_D$.

In this subsection we will describe into more details the types of structural heterogeneity addressed in this thesis.

The structural heterogeneity in ontologies can be resolved by considering the types of components and structures residing in the ontology graphs [44]. Methods that use such information rely on the intuition that elements of two distinct entities are similar when other elements adjacent to them are also similar [45]. Thus, we will set the following criteria [44] to decide if two entities in an ontology are structurally similar:

1. Their direct super-entities are similar.
2. Their direct sub-entities are similar.
3. All entities on the paths from the root to the entities in question are similar.
4. The entities in question have similar attributes/values.

We assume that the ***maximal common subgraph*** of both ontologies is not very small, which means that one hierarchy contains only a slightly smaller number of nodes compared to the other. This assumption is valid because in our framework ontologies are built on approximately the same sets of underlying concepts, with a particular domain in mind. The concept-to-concept correspondences are determined using the concept similarity applied on the set of the nodes.

For a class X in an ontology, it is relatively easy to determine if it is the union of several subclasses, $Y_1$,…, Yn by examining the knowledge in its description. For example in OWL (XML format), it is possible to read the `owl:unionClass` and `owl:subClassOf` XML tags. Similarly, it is possible to check if a class X is the intersection of two classes $Y_1$ and $Y_2$ by examining the `owl:intersectionOf` XML tag. We can also find equivalent classes for a particular class by examining the `owl:equivalentClass`, which is used create synonymous classes. In case of attributes of concepts, we use a similar procedure to check the similarity of properties in OWL. Similarity of the attributes is further used to determine the similarity of the concepts they define.

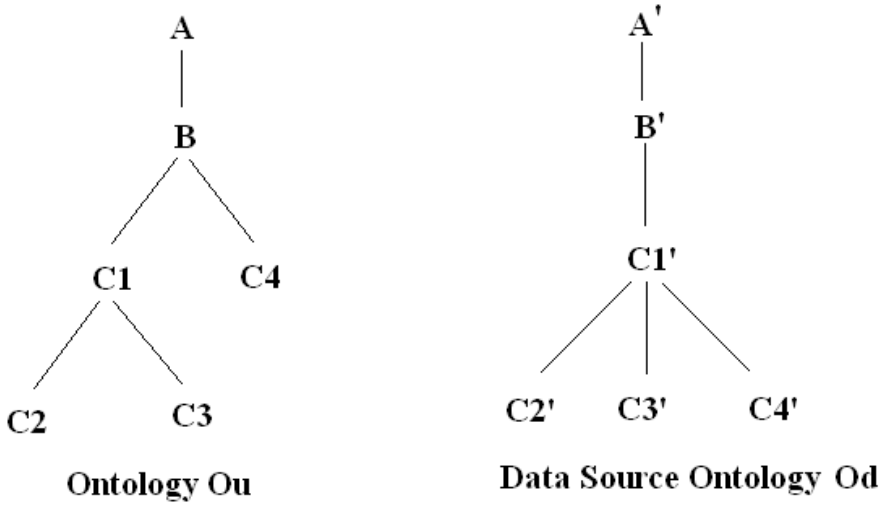
As we are mapping concepts in the user ontology to concepts in the data source ontologies, it is possible that similar concepts are placed at structurally dissimilar positions in the ontology graph. For example, it is possible that a concept C located at level k in the user ontology may be found at level l ($l \neq k$) in data source ontology. Although name-based and property-based comparisons will identify the similarity between the two concepts, it is necessary

to set up rules for mapping structurally heterogeneous concepts. There are two main cases of structural heterogeneity that we consider in this work, as described in what follows.

#### *4.2.1.1 Case 1*

We consider the hierarchical ontologies $O_U$ and $O_D$ in **Figure 4.1** and derive rules for mapping concepts in the user ontology to concepts in the data source ontology.

**Figure 4.1: Hierarchical ontologies $O_U$ and $O_D$**




We always start mapping from the leaves in the tree to the root. Let us assume that we have already mapped the classes C2, C3, C4 in $O_U$ to classes C2', C3', C4', respectively (based on lexical similarity). Thus, we have:

C2 ←→ C2'

C3 ←→ C3'

C4 ←→ C4'

From the ontology structure, we can see that:

C1' → C2' ∪ C3' ∪C4' in ontology $O_D$**(4)**

C1 → C2∪ C3 in ontology $O_U$**(5)**

B → C1∪C4 in ontology $O_U$**(6)**

→ C2∪C3∪C4

Also,

C2 ←→ C2'

C3 ←→ C3'

Thus, we can conclude that:

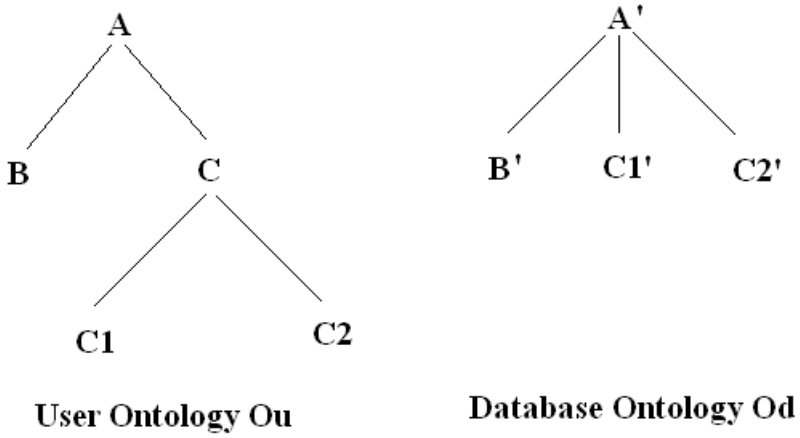
C1 ←→ C2' ∪ C3' (union of C2' and C3') **(A)**

B ←→ C1' **(B)**

#### *4.2.1.2 Case 2*

Next, we consider the hierarchical ontologies $O_U$ and $O_D$ in **Figure 4.2**

**Figure 4.2: Hierarchical ontologies $O_U$ and $O_D$**




We assume that we have already mapped classes C1, C2, B in $O_U$ to classes C1', C2', B' in $O_D$. We have:

C1 ←→ C1' → **(7)**

C2 ←→ C2' → **(8)**

B ←→ B' → **(9)**

From the structure of the Ontology $O_U$,

C → C1 ∪ C2 → **(10)**

A →B ∪ C → **(11)**

Therefore, we obtain:

C → C1' ∪C2' → **(C)**

A →B' ∪ C1' ∪ C2' → **(D)**

Thus, we will use rules (A), (B), (C), (D) inferred from the ontology structure to resolve structural heterogeneity when mapping between ontologies. We have designed a divisive procedure to identify structural heterogeneity of the types considered in Case 1 and Case2 above. Several hierarchical structures, which give rise to structural heterogeneity in ontologies like that in Case 1 and Case 2 above exist. Although other types of structural heterogeneity exist, we use only the two cases specified above. For the ontologies we considered, these are the most common types of structural heterogeneity. It should be noted that we are considering domain ontologies, with a large number of concepts (nodes in the hierarchy). Since the structures in Case 1 & Case 2 contain a maximum of 6 nodes, the same structures need to be applied repetitively to hierarchies to find structural similarity overall. It should also be noticed that Case 2 is a special case of Case 1, i.e. the hierarchical tree in Case 2 can be easily injected in the hierarchical tree in Case 1. Our mapping technique uses this procedure to search such structural differences between ontologies in an efficient manner.

## 4.3 Mapping Algorithm

To recapitulate, our mapping procedure aims to achieve the following tasks:

1. Resolve the lexical semantic heterogeneity between two ontologies.
2. Resolving the structural heterogeneity between two ontologies.

We can broadly divide the above two functionalities into two parts in the algorithm. First, we will compare the concepts or class names to determine their lexical similarity. In this phase, we compare class names and the names of the properties, which uniquely define these classes, to find similar terminologies. It is very important to check the properties (attributes) of the classes compared, as they define the 'has-a' relationships in the ontologies. The ‘has – a’ relationship in ontologies, define the characteristic attributes inherent to a concept. Consider the case in which the concept Movie_Name in the user ontology is defined as Movie _Title in the database ontology. If we are able to map the properties of the two concepts by comparing the range, domain, cardinality of the properties, we can conveniently say that the Concepts described by the properties are similar, in this case Movie_Name in the user ontology and Movie_Title in the database Ontology. In doing so, we will do a simple string comparison of the names of the classes under consideration. However if no such match is found, we will check to see if the

synonyms of the class names are similar. The synonyms of the class names can be found using JWI API in association with WordNet Lexical Database. If the algorithm is unable to find synonyms, which are "same" as the class names, then we will compare the hypernyms and hyponym of the class names to find any further match. If no such match is found at this point, we will declare the class "not mapable" lexically to any other concept or class in the second ontology. We then start checking the mapping the next class in the ontology.

In the second phase of the algorithm, we try to find structural similarities between classes in the two ontologies. We check the structural similarities by probing the positions of the classes in the two ontologies. This can be done by first checking the direct super-classes of the classes of interest in ontologies $O_U$ and $O_D$. We further check the direct subclasses of the classes of interest in ontologies $O_U$ and $O_D$. It is also necessary to check if the subclass of a class A from $O_U$ is either the same as the subclass of class A' from $O_D$ or at a lower level in the hierarchy, as compared to the subclass of class A' in $O_D$. Similarly, the super-class of a class A from $O_U$ is either the same as the super-class of class A' from $O_D$ or at a level greater than the super-class of class A from $O_D$.

In most ontology mapping techniques, the first and second phases of the mapping techniques are split, thereby resolving lexical and structural semantic heterogeneity separately. In most cases, the lexical heterogeneity is resolved first and then structural heterogeneity is handled in the later half of the mapping algorithm, by traversing the
XML tree for the ontologies in two different steps. As opposed to most other approaches, we resolve both semantic and structural heterogeneity together or at the same time, by handling each of them in a single traversal of the XML tree of the two ontologies, as follows.

It should be noted that at the end of the process we need to confirm that the subclasses of each class have been mapped. We start from the leaf nodes of the XML tree corresponding to the ontology $O_U$ (the nodes here are classes of the ontology). For each leaf node in $O_U$, a node with the same class name is sought for in the ontology $O_D$. When we find a node with same name in ontology $O_D$, we compare the properties of the nodes in order to confirm that they represent the same concept. When comparing the class attributes, a weight is assigned for each successful match.

The weights assigned are prioritized, i.e. a higher weight is allocated for a match of the domain, while the lowest weight is allocated for a comparison of the cardinality. We calculate

the total weights for each property and compare the total weight to a threshold value, which determines if the nodes are similar. If we are not able to find a node with same class name, we try to find the synonyms of the class name in ontology $O_D$. If we find a match, we next match their properties to determine if they are similar. If a synonym match is not found, we find hypernyms and/or hyponyms for the class name and search for the same in the ontology $O_D$. When a hypernym and/or hyponym match is found, we repeat the same process for comparing properties. If a semantic match is not found in any of the three steps above, we will move to the next leaf in the XML tree. When a semantic match is found, we will check the structural similarity. To do so, we will first check the direct parent of the node. If a match is found we will allocate a weight of 0.2, otherwise a weight of 0 is allocated. We will further check the node's direct child. If a match is found we will allocate a weight of 0.2, otherwise a weight of 0 is allocated. We further check the predecessors of the node. For each predecessor match, we allocate a weight of 0.1; otherwise we allocate a weight of 0. It is not necessary to check the children, successors of the classes, as we start from the leaf nodes. At this point, we have checked the direct parent, direct child of the class. We have also checked the predecessors and successor of each class, to determine if a structural match for node has been found in the other ontology. We calculate the total weights for each structural comparison, and compare it to a threshold value. If the total weight of any class is greater than a threshold value, the class can be termed structurally similar.

## 4.4 Resources Used by the Mapping Procedure

### *4.4.1 JDOM (Java Document Object Model)*

We use JDOM to represent ontologies as tree structures and to find structural dissimilarities in the two ontologies, if any. JDOM is also useful when mapping concepts at the lexical level.

#### *4.4.1.1 JDOM Description and Features*

JDOM is an open source, tree-based, pure Java API for parsing, creating, manipulating, and serializing XML documents, developed by Brett McLaughlin and Jason Hunter in the Spring of 2000[46]**.** Although JDOM is similar to the World Wide Web Consortium's (W3C) DOM, it

has an alternative document object model, which was not built on DOM or modeled after DOM. The main difference is that while DOM was created to be language-neutral and initially used for JavaScript manipulation of HTML pages, JDOM was created to be Java-specific and thereby takes advantage of Java's features.

JDOM represents an XML document as a tree composed of elements, attributes, comments, processing instructions, text nodes, CDATA sections, etc[46][47][48[49]. The XML document represented as a tree and the components of the tree are available to the developer at any time.

It should be noted that JDOM itself does not include a parser. Instead it depends on a SAX parser with a custom ContentHandler to parse documents and build JDOM models from them. JDOM can build a new XML tree in memory. Data for the tree can come from a non-XML source like a database or from literals in the Java program. Once a document, created from scratch or parsed from a stream, has been loaded into memory, JDOM can modify the document. A JDOM tree is fully read-write. All parts of the tree can be moved, deleted, and added to the tree, subject to the usual restrictions of XML.

#### *4.4.1.2 JDOM Advantages*

JDOM consistently uses the Java coding conventions and the class library. For example, all primary JDOM classes have `equals()`, `toString()`, and `hashCode()` methods. They all implement the `Cloneable` and `Serializable` interfaces. JDOM checks all the data in the newly created XML documents for well formedness, when it creates XML documents from scratch. JDOM allows one to serialize it back out to disk or onto a stream as a sequence of bytes, when finished working with a document in memory. JDOM provides numerous options to specify the encoding, indenting, line end characters, and other details of serialization.

#### *4.4.1.3 JDOM Disadvantages*

JDOM represents the different kinds of nodes in the tree by concrete classes rather than interfaces. It may be noted that JDOM cannot handle documents larger than available memory.

JDOM loads the entire document tree into memory and stores it there for future use. JDOM is more memory efficient than some DOM implementations and less efficient than others. JDOM presents a logical model of XML documents. It cannot tell which physical entities a particular element came from. It treats the document as a logical whole. It cannot tell whether a character was input literally or with a character reference (a limitation it shares with DOM and SAX). It does not know the original character encoding of the document. In short, it cannot guarantee byte-for-byte faithful round trips. Thus it is possible that parsing a document with JDOM and then immediately writing it back out again may create a subtly different document, though it should still contain the same basic information. JDOM does not provide any equivalent to the traversal package in DOM. However it does not take a huge effort to write your own tree-walking code.

### *4.4.2 WordNet*

#### *4.4.2.1 WordNet Description*

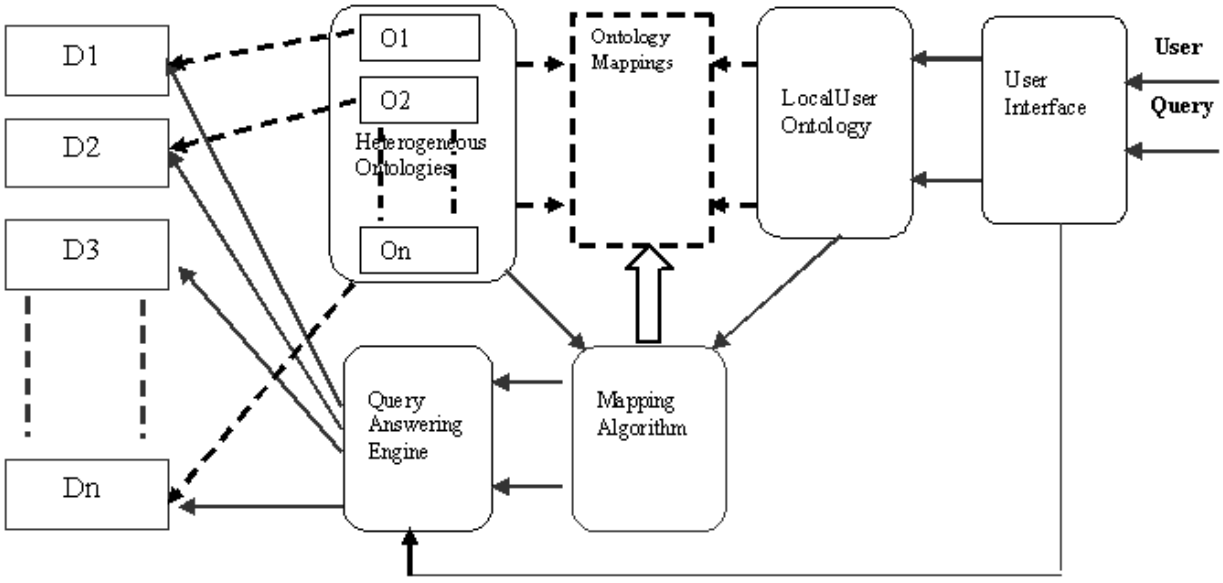
WordNet® is a free and publicly available lexical database of English which was developed under the direction of George A. Miller [16]. WordNet's source files are written by lexicographers. WordNet is a result of a variety of lexical and semantic relations used to represent lexical knowledge. The words in English language are represented as synsets, which are lists of synonymous words that can be used in some context. Synsets include nouns, verbs, adjective and adverbs, grouped separately, each expressing a different concept. WordNet's structure makes it a useful tool for computational linguistics and natural language processing [16].

#### *4.4.2.2 MIT Java WordNet Interface*

The **MIT Java WordNet Interface (JWI)** [50] is a freely available, easy-to-use Java library used to interface with the WordNet electronic dictionary. Although the interface does not include any GUI element, it allows the user to directly call procedures/functions to retrieve index words, synsets, and morphological exceptions from the WordNet data files. It can also be used for browsing the WordNet lexical database, by following lexical and semantic pointers.

## 4.5 System Architecture

**Figure 4.3: System Architecture**




### *4.5.1 User Queries and Query Processing Engine*

**Figure 4.3** describes the architecture of the system that was designed to extract and integrate knowledge from multiple heterogeneous data sources. The user interface is used for direct interaction of the users with the system, thereby making the application user-friendly. User can use the interface to send queries to the system, using a simple standard query language. Thus, we ensure that the user does not need to have detailed "know-how" of complex querying languages, such as SQL, to query the heterogeneous databases.

The diagram in **Figure 4.3**, shows a set of multiple heterogeneous ontologies $O_1$, $O_2$… $O_n$. These ontologies provide a formal representation for the heterogeneous data sources $D_1$, $D_2$,…, $D_n$. In principle, the data sources under consideration can be flat files, relational databases, object oriented databases, etc. In this thesis, we only consider relational databases.

Each data source can be seen as a semantic resource that can be transparently queried by users, using a simple query language, through the means of a query answering engine. The user is assisted by an ontology $O_U$, which represents the user's perspective of the concepts in the multiple data sources or the domain under consideration. When the user first registers with the system, he or she needs to specify the user ontology. The mapping algorithm maps the concepts in the user ontology to concepts in the heterogeneous data source ontologies, resolving the lexical and structural semantic heterogeneity in most cases. As mentioned above, we assume that all ontologies are written in OWL (XML format only).

The user can post a set of well-defined queries to the system, formulated in terms of his/her own ontology. The query answering engine decomposes the user query into sub queries that can be answered by the data sources in the system, maps them to the data source ontologies by using the set of mappings and finally transforms them into SQL inline queries. It is necessary to transform the user queries to database queries since the database does not understand the user query language. The query processing engine sends these queries to each of the databases depending on the mapping information provided by the algorithm. Each of the databases implements the queries, returning result sets which are sent to the user interface for display to the user. We chose to use ORACLE/SQL inline queries by taking into consideration the complexity of the queries which need to be formulated, and the fact that the database may or may not consist of relational tables. It would be very difficult to formulate other queries in the present form of the implementation, since we also need to check the primary and foreign keys for tables.

#### *4.5.1.1 SQL Inline Queries*

The inline view/query is a construct in Oracle SQL where you can place a query in the SQL FROM, clause, just as if the query was a table name. A common use for in-line views in Oracle SQL is to simplify complex queries by removing join operations and condensing several separate queries into a single query [51].

In Oracle SQL, it is quite difficult to compare two result sets that are summed together in a single query where specific values must be compared to a summary. Without the use of an in-line view, several separate SQL queries would need to be written, one to compute the sums from each view and another to compare the intermediate result sets. The other possible way to achieve a similar result is to construct queries for each view and then to create a temporary table to store the record set. This temporary table can further be queried to get the necessary result. Thus

instead of spending a lot of I/O and disk space, or for that matter disk access as in the case of a temporary table, it is possible to get the same result using an inline query. We also avoid the necessity to create joins between tables, to extract information from multiple tables. In short, if one only needs the data to join to other queries, it may be good to try using an inline view to conserve resources.

#### *4.5.1.2 User and Data Source Ontologies*

In this architecture, the data source ontologies are designed in a comprehensive and task-neutral fashion, without regard to the user and the various applications a user may be interested in. The user ontology, on the other hand, is custom crafted to support a specific customer and a specific set of tasks. As a result, the data source ontologies can be reused for multiple applications, assuming suitable integrating user ontology and a corresponding set of translation rules is developed.

We used Protégé 3.3 to prepare the user and data source movie ontologies in our system. Protégé is a free, open source ontology editor and knowledge-base framework. It is Java based extensible and platform-independent environment. Its plug and play environment makes it easy to develop, integrate and query ontologies [52]. According to [52], the Protégé ontology editor enables users to:

- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties, and SWRL rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoner, such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup.

#### *4.5.1.3 Oracle DBMS*

In our system, we use multiple Oracle databases as heterogeneous data sources. It is believed that Oracle is one of the most stable, secure and robust relational database management systems. The portability of the Oracle database system over a range of operating platforms is crucial to the application. Oracle also provides users with certain system tables, which can used extensively to find information about table or schema metadata. This makes it easier for the user

to fetch data form the database about the database schema. The Oracle database has been particularly chosen, to allow us to use the system tables like dba_col_constraints, dba_constraints, dba_tables, to extract meta-data about the tables, columns and their constraints. Thus we do not have to write separate scripts to determine the primary and foreign keys linking the relational tables. It should also be noted that the application under consideration is targeted at medium to large datasets, which makes Oracle RDBMS a better choice for this application.

The data in an ontology-extended database is such that the table names are the names of the classes at level 1 in the XML tree of the data source ontology. The attribute names (labels) are used as column names for the tables. The leaf nodes in the XML tree are values in the database (together, forming instances).

#### *4.5.1.4 Database Assumptions*

Although we use Oracle databases to serve the purpose of heterogeneous data sources, we make certain assumptions that are valid for any other relational data source that might be used in our application, as follows:

1. Assume, without loss of generality, that all relations in the schema are in 3NF.
2. A relational database schema is a tuple (ET, tab, col, dtype, pk, fk), where
   ET = set of entity tables names associated with the data source ontology
   tab = table name of the tables comprising the relational database,
   col = column names in the tables,
   dtype = data types of the columns in the relational tables,
   pk= primary key in a table,
   fk = foreign key in a table.

We further assume that:

- Each entity table consists of rows of instance specific data corresponding to the entities described in the associated data source ontologies.
- Each entity table has a unique name. No two entity tables can have the same name.
- No two columns have the same name; each column name is unique.

- Each datatype ***dtype*** is a predefined RDBMS datatype, specifying a value range of the relevant instance data. For each entity table $t$, there is a finite nonempty set col($t$) of *column* names; each $c \in$ col($t$) has an associated datatype, denoted datatype($c$) $\in DT$.
- Each relational table has one and only one primary key.
- Each relational table may have one or more foreign keys.
- Subof $\subseteq t \times t$ is a binary relation over $ET$ that models an *inheritance relation* between two entity tables. For some $t$, $r \in ET$, subof($t$,$r$) is satisfied

iff $\exists$ fk($t$, $r$) $\in$ col($t$) such that either

fk($t$, $r$) = pk($t$) (*single inheritance*)

OR

fk($t$, $r$) $\in$ pk($t$) (*multiple inheritance*).

Here $t$ is a *subentity table*, $r$ is a *superentity table*, and all the related tables form a *generalization hierarchy of the entity tables*

#### *4.5.1.5 Why RDBMS?*

A relational database is a collection of data items organized as a set of tables from which data can be accessed without having to reorganize the database tables. The application is targeted at medium and large datasets or data sources. It will be unreasonable to believe that the data in such datasets could be compressed in a single table. This necessitates the use of a stable, robust RDBMS.

#### *4.5.1.6 Preprocessing of RAW Movie Data*

The 'raw' data for the underlying database was downloaded from the IMDB movies website (www.imdb.com/interfaces).The original data was in the form of LIST files with each file containing data pertaining to a particular aspect of movie making/marketing. Each LIST file consisted of data represented in the form of rows and columns. A data row/record was represented as a single line of data in the file, with each column of data being separated by a space character. The raw data was incomplete, inconsistent and in some cases redundant or incorrect, making it difficult to determine which data to use. In this thesis, we have considered

all complete, consistent records of data, removing any redundant or incorrect data. This 'raw' data was converted to 'usable data' by parsing the data using C++ code. The C++ code parsed the data in such a way that only "usable" data was made available in the output file. An example would be the "business" list, which had additional data not required in the database and representation not available in the mapped Ontology. The "business" LIST file was parsed in such a way that only the necessary fields namely, name of the movie, money made by the movies in different countries and the release date associated with the movie were extracted to form the usable "business" list text file. Fundamentally, the C++ code removed all the unwanted data to convert the data in the raw LIST file into usable text files. Each record in the text data file was represented in a single line and the end of each record is indicted by carriage return character and a line feed character. Each column in the record is separated by comma. In some cases, the column data was enclosed in double quotation characters to indicate the beginning and end of the string.

#### *4.5.1.7 Loading Data into RDBMS*

The choice of Oracle DBMS to serve as heterogeneous data source also depended on tools available for effective import, export of data from external data sources; e.g flat files. Oracle also allows us to create multiple users, thereby allowing multiple users working on similar domains to connect to the Movies Database.

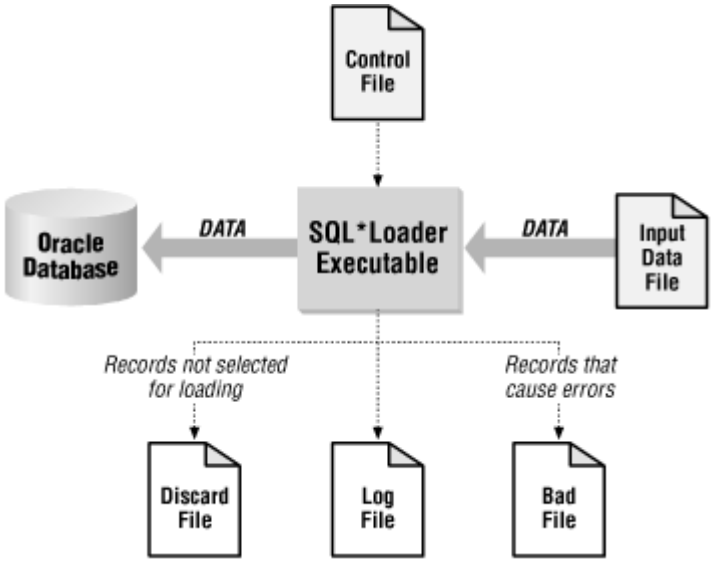
Oracle provides the SQL Loader utility for importing and exporting data from external sources to the database. We will briefly describe the Oracle SQL Loader utility and how it has been used in this thesis implementation.

#### *4.5.1.8 SQL* Loader*

SQL*Loader is an Oracle-supplied utility that allows to load data from a flat file into one or more database tables. The basis for everything done with SQL*Loader is a file known as the *control file*. The SQL*Loader control file is a text file which provides a description of the data to be loaded into the database. It may or may not be the data of an entire table, since we may load data in multiple tables or selectively load data in one or more columns in a table. Thus we may say that the control file is used to tell SQL*Loader, which database tables and columns should receive the data that you are loading.

The external flat file consisting of the data to be loaded in the database is called the flat file while the control file describes the data contained in that data file. There is another important file called the log file, which logs the operation of importing or exporting data to or from the database. The *log file* is a record of SQL*Loader's activities during a load session. It is necessary to review the log file after a load to be sure that no errors occurred, or at least that no unexpected errors occurred.

**Figure 4.4: SQL Loader Architecture**




The diagram in **Figure 4.4** [51] gives an overview of the Oracle SQL* Loader operation in conjunction with the Oracle database, using the control file, data file and log file to effectively load data to the database. The SQL*Loader executable is invoked, which points the load process to the control file created for loading the data. SQL*Loader reads the control file to get a description of the data to be loaded. Then it reads the data file and loads the input data into the database. We can say that SQL*Loader is a very flexible utility to load and unload data from the Oracle database.

We have briefly discussed the resources used in the design and development of the system. We have also explained the system architecture in detail, to evaluate mapping of multiple heterogeneous ontologies and querying of the underlying databases. The differences between the original Naïve Bayes algorithm and the ones implemented in the system have been discussed, thereby exploiting the heterogeneous data sources to build predictive models. We will

describe the experimental setup for evaluating the querying of heterogeneous data sources in the next chapter.

# CHAPTER 5 - Experimental Design

This Chapter describes the details of the experiments conducted to comprehensively evaluate the ontology mapping procedure and the querying of heterogeneous data source, as discussed in Chapter 4. A set of experiments was specifically designed to investigate the mapping process in the architecture. A different set of experiments, using the same architecture, have been designed to investigate the querying of heterogeneous data sources.

## 5.1 Evaluation Goals

Our first goal was to evaluate the performance of the mapping algorithm we have designed. We will use the response time analysis to determine the performance of the mapping algorithm.

## 5.2 User and Database Ontologies

As specified in Chapter 4, the data source ontologies are designed in a comprehensive and task-neutral fashion, without regard to the user and application, while the integrating user ontology, on the other hand, is custom crafted to support a specific customer and a specific set of tasks.

Our domain for querying multiple heterogeneous data sources is the movie domain. We have custom-designed a user ontology for the movie domain, which presents the user's perspective of the domain and supports a specific set of tasks. The data source ontologies have been constructed with regard to the data present in the oracle database, so that they better represent the structure of data in the databases. Thus, these ontologies are expected to consist of lexical and structural heterogeneity as compared to the user ontology.

### *5.2.1 Case 1: Single User Ontology, Single Database Ontology*

In this first experiment, we consider a user ontology and a single database ontology. The primary focus is to evaluate the performance of querying in the presence of the mapping algorithm, by determining the time needed to map the user's query to a database query, together

with the time to answer the mapped query. In particular, we want to determine how the querying time varies with the number of attributes (describing data in the same table or several tables) in the query. We will evaluate the mapping/querying times, by querying the database as follows:

- Query a single attribute from a single table.
- Query single attribute with two attribute values from the single table.
- Query two attributes from different tables.
- Query single attributes with three attribute values from the single table.
- Query three attributes from three different tables.
- Query multiple attributes from multiple tables.

We will separately query the Business table, to check the functional mappings in the ontologies, in particular the currency conversion. We separately use one or more attributes with the currency attribute in the Business table, to evaluate the query performance for the functional mappings. Since we have caused a dynamic mapping of the functional concepts, in particular currencies, it is important to evaluate the performance of the aggregation function, which internally applies the conversion function.

### *5.2.2 Case 2: Single User Ontology, Multiple Data Source Ontologies*

In this case, the primary focus is to evaluate the time needed for query mapping and query answering from *multiple* heterogeneous data sources. There is a single user ontology and two heterogeneous data source ontologies providing a formal representation of their underlying data sources.

We will evaluate the query mapping process for two lexically and structurally heterogeneous data source ontologies. The underlying heterogeneous databases are two Oracle databases, set up on the same Oracle server. We will determine the mapping time, query retrieval times and the difference in the queries as follows:

- Query a single attribute from a single table in the two heterogeneous databases.
- Query single attribute with two attributes values from single table in the two heterogeneous databases.
- Query two attributes from different tables in the two heterogeneous databases.

- Query single attribute with three attributes values from single table in the two heterogeneous databases.
- Query three attributes from three different tables in the two heterogeneous databases.
- Query an attribute which shows lexical semantic heterogeneity in the two databases.
- Query multiple attributes which shows lexical semantic heterogeneity in the two databases.

Again, we will separately query the Business table, to check the functional mappings in the ontologies, in particular the currency conversions. This is specifically done to check if the functional mappings, which cause internal currency conversion and additional queries report a change in the mapping and query retrieval time.

### *5.2.3 Case 3: Single User Ontology, Multiple Database Ontologies, with Missing Leaf Nodes*

In this case, the primary focus is to evaluate the mapping and querying of multiple heterogeneous data sources, to check if it correctly detects missing nodes in one data source and uses the other ontology and corresponding data source to give a weighted valued for the missing node in the ontology, when queried by the user.

Consider the case in which a leaf node or concept of Action 1 (type of action movies, as categorized by data source ontology $O_U$) is absent in Ontology $O_{D1}$. Similarly, another concept of Action 2 (type of action movies, as categorized by data source ontology $O_U$) is absent in the ontology $O_{D2}$. Let us further assume that Action 1 and Action 2 have super class Action, which has a super class Genre in ontology $O_{D1}$. However in ontology $O_{D2}$, class Genre has subclass Action. However, the subclasses Action1 and Action2 are absent in ontology $O_{D2}$. The user ontology $O_U$ has structure similar to ontology $O_{D1}$.

Because the user is unaware of the absence of the concepts Action1 and Action2 in ontology $O_{D2}$, he/she queries the data source $O_{D2}$ for the number of movies with genre Action1. The system which has both data source ontologies available, will map the user ontology $O_U$ to the data source ontologies $O_{D1}$ and $O_{D2}$. The user ontology creates the following mappings,

**Table 5.1: Concept Mappings between user ontology and data sources ontologies $O_{D1}$ and $O_{D2}$, for missing leaf nodes.**

| **Mapping of $O_U$ to $O_{D1}$**<br>**$O_U \leftrightarrow O_{D1}$** | **Mapping of $O_U$ to $O_{D2}$**<br>**$O_U \leftrightarrow O_{D2}$** |
|---|---|
| $O_U$:Genre $\leftrightarrow$ $O_{D1}$:Genre | $O_U$:Genre $\leftrightarrow$ $O_{D2}$:Genre |
| $O_U$:Action$\leftrightarrow$ $O_{D1}$:Action | $O_U$:Action$\leftrightarrow$ $O_{D2}$:Action |
| | $O_U$:Action$\leftrightarrow$$O_{D2}$:Action1 $\cup$ $O_{D2}$.Action2 |

The system, at this point will know that the class Action1 and Action2 are not available in ontology $O_1$. The system makes an assumption that the proportion of Movies with genre, Action1 and Action2, is the same in both $O_{D1}$ and $O_{D2}$.The system then finds the number of movies with genre Action1 and number of movies with genre Action1 and Action2, i.e. number of movies with genre Action in the data source corresponding to data ontology $O_{D2}$. It finds the weighted percentage of movies with genre Action1, in movies with genre Action. It further finds the total number of movies with genre Action in Ontology $O_{D1}$. It applies the percentage distribution of movie with genre Action1 in Ontology $O_{D2}$ to this number, to obtain a weighted value of the number of movies in Ontology $O_{D1}$ with genre Action1.

We will evaluate the mapping time, query retrieval times and the difference in the queries as follows:

- Query the attributes missing in one of the two ontologies.

Limitation: We can only query an attribute from a single table in this case, that of the missing leaf node in the ontologies. It is not possible to calculate the distribution of a missing attribute with another attribute, which may or may not be present in the data source.

### *5.2.4 Case 4: Naïve Bayes Algorithm Applied to Single User Ontology, Single Database Ontology*

In this case, we will use the Naïve Bayes algorithm, to check the validity of the query processing engine and the mapping process, inadvertently, with a single user and database ontology. The formulas used for the relational Naïve Bayes algorithm are given in Section 2.5.

The user poses a learning (or prediction) query, in which the last attribute is assumed to be the class attribute. The system will use the user and data source ontology to map the classes and accordingly formulate queries for estimating probabilities needed for learning. Since we use two different approaches to inference, we will return two different values.

As only one data source is assumed in this case, the system uses the data source available both as training data to estimate probabilities needed for learning and as test data to evaluate the accuracy of the learned classifier.

### *5.2.5 Case 5: Naïve Bayes Algorithm Applied to Single User Ontology, Two Database Ontologies*

We will repeat the same procedure as in Case 5, except that there will be two data sources used, with their underlying ontologies. The difference in this case, is that the system queries the data source 1, learns from the knowledge acquired from data source 1 and uses this knowledge to acquire knowledge from the second data source.

# CHAPTER 6 - Results

This chapter documents the results from the experiments conducted on the system, as described in Chapter 5.We will investigate the performance and efficiency of the mapping technique to the querying of heterogeneous data sources. We will also examine the querying of the heterogeneous data sources by inspecting the database queries, constructed by the query processing engine. We will study database fetch times to check if the number of records returned in the result set, or the number of attributes in the query make a difference to the query processing time by the database itself.

## 6.1 Querying a Single Data Source, Using a Single User and Database Ontology

This section presents the performance of the mapping technique used in querying multiple heterogeneous data sources. We wanted to check if the mapping times for the concepts varied with the number of attributes being queried, or the number of records fetched from the data sources changed the query times. Therefore, we have performed experiments on the system by querying different number of attributes from one or multiple tables. The mapping and query times in each of the cases have been averaged out from a series of 5-6 observations. Thus we have made sure that we account for any minor network delays or I/O discrepancies which may exist in the timing values. The mapping and query times are expressed in milliseconds.

**Table 6.1.1** shows the mapping and query time, records fetched for different queries on a single attribute in a single table. It is evident that there is not particular association between the mapping times and the number of concepts queried.

However, it can be seen that the query times increase with an increase in the number of records fetched for all queries, with the exception of the first record which has the least number of records. It can also be observed that the mapping times for all these records differ from each other by less than 1.0 millisecond with the exception of the third record.

The following are the results of querying a single attribute in a single table

**Table 6.1.1: Results for querying a single attribute in a single table**

| Table Name | Mapping Time (msec) | Query Time (msec) | Records fetched |
|---|---|---|---|
| Country (Romania) | 15.8 | 143.8 | 1354 |
| Genre(War) | 31.8 | 122 | 2001 |
| Language (Hindi) | 47 | 146.8 | 6622 |
| Country (India) | 34.4 | 156 | 14899 |
| Genre (Action) | 31.6 | 162.6 | 15062 |
| Country (USA) | 31.4 | 314 | 116822 |
| Language (English) | 31 | 387 | 154221 |

**Table 6.1.2** shows the mapping and the query times for queries, when two attributes from single table. These observations are very critical to evaluate the efficiency of the query processing engine, since the query processing engine also queries certain system tables in the Oracle DBMS to check the constraints in the tables. We can thus ascertain if this phase of query processing take a considerable amount of time. In fact, during the mapping phase, the system does a sanity check on the attributes queried in the user query to validate the existence of the attributes in the data source tables. Thus, we ensure that before we actually proceed to the query processing phase, we have checked the existence of the attributes. In this stage we are querying two attributes from two different tables, so we may expect the timings for the queries to increase.

However it has been observed that although the mapping times have not increased, the query times have shown a slight increase. It may also be observed the query times have increased with the number of records fetched, with the exception of record 2.

The following are the results of querying two attributes from two different tables.

**Table 6.1.662:Results for querying a single attribute with two attributes values in a single table**

| Table Name | Mapping Time (msec) | Query Time (msec) | Records fetched |
|---|---|---|---|
| Country (Romania, UK) | 15.8 | 290.6 | 1354 |
| Genre (Comedy, Classic) | 31.33 | 318.33 | 65895 |
| Genre (Action, Adult) | 15.25 | 308.25 | 66936 |
| Country (India, USA) | 15.75 | 398.25 | 131329 |
| Language (English, Hindi) | 15.67 | 432.33 | 160363 |
| Language (Spanish, English) | 15.4 | 540.4 | 173103 |

**Table 6.1.3 Results for querying two attributes from multiple tables**

| Table Name | Mapping Time (msec) | Query Time (msec) | Records fetched |
|---|---|---|---|
| Genre, Country | 15.8 | 353 | 6007 |
| Country, Language | 24.6 | 274.8 | 7135 |
| Genre, Language | 15.33 | 242.16 | 7550 |
| Genre, Language | 27.46 | 633.71 | 37647 |
| Language, Country | 15.6 | 1590.6 | 100493 |

The results for querying two attributes from two different tables are presented in **Table 6.1.3**. There is no particular relation that we have been able to establish for any of the records. However it is visible that the mapping times have not changed much, with the exception of one

particular record (record 5), while the query times have increased from the case in which two attributes from the same table were queried.

This can be attributed to the fact that the query processing engine collects meta-data about the table and their columns to construct the query. It would obviously take more time collect the metadata about two tables, as compared to one table.

**Table 6.1.4** gives the results of querying three different attributes from the same table. It can be seen again, that the query times of the records increase with increase in the number of records fetched from the table. The mapping times do not show any dependency on the number of records fetched by the query from the data source.

**Table 6.1.4: Results for querying single attribute with three attributes from a single table**

| **Table Name** | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** |
|---|---|---|---|
| Language | 15.4 | 272 | 8878 |
| Country | 31.66 | 292 | 15884 |
| Genre | 31 | 387.4 | 68477 |
| Genre | 16.5 | 391 | 79900 |
| Country | 31.25 | 425.75 | 84068 |

The following are results of querying three attributes from three tables,

**Table 6.1.5:Querying three attributes from three tables**

| **Table Name** | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** |
|---|---|---|---|
| Genre, Language, Country<br>War, English, UK | 46 | 233.2 | 257 |
| Genre, Language, Country<br>Comedy, Spanish, Mexico | 31.33 | 333.5 | 850 |
| Genre, Language, Country<br>Romance, India, Hindi | 31.4 | 315 | 948 |
| Genre, Language, Business<br>India, Marathi, Money | 31.5 | 218.25 | 0 |
| Country, Language, Business<br>Romania, Romanian, Money | 31.75 | 199.25 | 0 |

**Table 6.1.5** summarizes the results for three attributes being queried from three different tables. It may be observed that there is no particular relation between the number of query and mapping times with the records fetched in the queries.

The following are results of querying multiple attributes from one or more tables,

**Table 6.1.6: Results of querying multiple attributes from one or more tables**

| Table Name | Mapping Time (msec) | Query Time (msec) | Records fetched |
|---|---|---|---|
| Genre, Language, Country (Drama, English, UK) | 42.75 | 16941.5 | 2866 |
| Genre (Drama, War) | 31.66 | 255 | 3440 |
| Genre, Language, Country (Adventure, English, UK) | 31.75 | 24144.5 | 4754 |
| Genre, Language, Country (Short_Film, English, UK) | 31 | 37121.33 | 5335 |
| Genre (Sport, Short_Film) | 15.25 | 342.5 | 85830 |

This table summarizes the results for querying multiple attributes, which further derive other attributes (i.e. concepts which are not leaf nodes). These queries are very critical to understanding the mapping timings, considering that it is the mapping phase which maps the leaf nodes in the user ontology to the leaf nodes in the data source ontologies. It may be seen that the concepts which derive 3-4 leaf nodes (or child concepts) and query multiple tables have a very high query time. An example would be record 5, which fetches more records from the same table, Genre as opposed to record 4, which fetches lesser number of records from multiple tables. In the 4$^{th}$ record, the query is fetching records for more attributes than the 5$^{th}$record, but the difference in the query times is very large.

The mapping of the concepts takes almost the same time, as any of the cases described above. This only shows the efficiency of the mapping technique used in the system. The query times for the queries have increased due to querying of multiple tables and multiple concepts (more than 3).

There are several reasons for the variable query times in the cases above. The most important ones of all these are I/O associated with the local computer system on which the database servers are installed, the performance of the network used for querying the data sources. The other reason which are a major factor in the variable query times, and is specific to this system, is the structure of the oracle database system itself.

Oracle itself has an architecture which minimizes the number of 'logical' and 'physical' fetches from the actual data repository.

The oracle architecture has a memory structure called the shared pool, which has components such as the library cache, data dictionary cache. It also has other memory structures like database buffer cache, redo log buffer, which are very critical to the performance of our system. The actual data is cached in the database buffer cache in the form of data blocks for efficiency and performance of the system.

The oracle DBMS targeted at middle and large databases, stores all the data in the physical memory. When a user queries the database, the architecture first checks if the data is available in the oracle database buffer cache. If the server process finds the necessary data blocks in the cache, it returns the records to the user directly. This is a called a "logical fetch". However if the server process does not find the records in the cache, it performs a 'physical fetch' by getting records from the physical memory, which leads to disk I/O. This data which is in the form of database blocks, which are stored in the database buffer cache. The list of blocks which are brought to the cache from the physical memory are also stored in a LRU (least recently used) list, which manages the list of block being used by the database. This is a very brief explanation, as to how the oracle database actually implements a query on table. It is evident that the size of the shared pool and the database buffer cache are very critical to our system. If the size of the shared pool and the database buffer cache were less than the optimal size required, the number of physical fetches is much more, leading to higher values of query times. At the same time, a large sized shared pool and database buffer cache will lead to unnecessary data buffers being cached in the database buffer cache and will lead to unnecessary use of cache space. We have used the size of the shared pool and the database buffer cache as 'advised' by the Oracle DBMS when creating the database.

The process described is very brief and the detailed process is much vaster and takes a few milliseconds to implement. The process described above plays a very important role in the query times of the system being tested.

It may also be noted that a physical fetch is performed every time, a different attribute from the same/different table is queried. The possibility of a user posing a query with the same attribute multiple times repeatedly is very scarce as compared to user querying different aspects of the multiple tables. Thus we may say that the above factor plays a very important role in influencing the query times of the databases.

It may also be noted that similar architecture exists for most database systems targeted at middle and large database systems. So we may conclude that such discrepancies in the query times will also in other database systems targeting medium and large database systems.

## 6.2 Querying Multiple Data Sources, Using a Single User and Multiple Database Ontologies

This case is particularly useful to evaluate the mapping of semantically and structurally heterogeneous concepts and querying the underlying the data sources.

As in the above case, we have taken results for different number of attributes querying one/multiple tables in two data sources. This will help us better understand the performance of the mapping technique and query processing in the system. This will also reveal the differences in the mapping and query times from two different databases, depending on the records in the tables, the structure of the tables.

The mapping and query times in each of the cases have been averaged out from a series of 5-6 observations. Thus we have made sure that we account for any minor network delays or I/O discrepancies which may exist in the timing values. It may be noted that the mapping and query times are in milliseconds.

**Table 6.2.1** shows the summary of the mapping and the query times for a single attribute being queried from a single table in two databases.

**Table 6.2.1: Querying a single attribute in a single table (two databases)**

| Tables Queried | Data Source 1 Orcl | | | Data Source 2 Orcl2 | | |
|---|---|---|---|---|---|---|
| | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** |
| Genre (Horror) | 1398.5 | 946.5 | 7386 | 1562.5 | 922 | 7151 |
| Country (Spain) | 5200.67 | 1015.5 | 8460 | 5290.75 | 938.5 | 8460 |
| Genre(Crime) | 1625 | 934.5 | 9784 | 1570.25 | 903 | 9245 |
| Country (India) | 6484.25 | 958.3 | 14899 | 6585.75 | 934.4 | 14899 |
| Language(French) | 2706.4 | 909.6 | 17185 | 2640.8 | 878 | 17185 |
| Genre(Adult) | 2411.33 | 918 | 27483 | 2260.33 | 902.25 | 27351 |
| Language(Hindi) | 3031 | 903 | 36910 | 1395.2 | 896.6 | 36304 |
| Country(USA) | 3124.75 | 1184.2 | 116822 | 3415.6 | 1080.66 | 116822 |
| Language(English) | 1443.8 | 1222 | 154221 | 1145.66 | 1168.8 | 154221 |

It can be seen from **Table 6.2.1**, that there is no particular relationship between the number of records fetched and the mapping and query times for the queries. However, it is evident that the query times for both the data sources are in the same range (900 milliseconds – 1200 milliseconds), for any number of records fetched. Also the mapping and query times for both the data sources, for any query, match each other by a factor of not more than 100 milliseconds, in the two data sources.

**Table 6.2.2** presents a summary of the mapping times, query times and records fetched for querying with two attributes from a single table

**Table 6.2.2: Querying single attribute with two attribute values in single table (two databases)**

| Tables Queried | Data Source 1 Orcl | | | Data Source 2 Orcl2 | | |
|---|---|---|---|---|---|---|
| | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** |
| Genre (Crime Horror) | 983.25 | 875.5 | 16710 | 1134.25 | 862 | 15979 |
| Language (English, Spanish) | 1463.8 | 1078.2 | 20890 | 1593.6 | 1085.8 | 20890 |
| Language (French, Italian) | 1065 | 841.5 | 25505 | 1101.5 | 828.25 | 25505 |
| Genre (Adult Sport) | 1411.25 | 933 | 30152 | 1444.25 | 874.75 | 29966 |
| Country(UK, Canada) | 11825.83 | 1167.33 | 30979 | 11549.66 | 986.83 | 30979 |
| Genre (Adult, Horror) | 1322 | 877 | 34546 | 1085.4 | 853.6 | 34206 |
| Genre(Adult, Crime) | 2481.5 | 1023.5 | 36910 | 2415.8 | 940.6 | 36304 |
| Country (USA, UK) | 1422 | 1023 | 132025 | 1350.5 | 1073.25 | 132025 |
| Language(English, French) | 3661.5 | 1251 | 168667 | 3707.5 | 1174.5 | 168667 |

It can be seen from the above table, that there is no particular relationship between the number of records fetched and the mapping and query times for the queries. However, it is evident that the query times for both the data sources are in the same range (900 milliseconds –

1200 milliseconds), for any number of records fetched. Also the mapping and query times for both the data sources, for any query, differ from each other by a factor of more than atleast 40 milliseconds, in the two data sources.

**Table 6.2.3** summarizes of the mapping times, query times and records fetched for querying with two attributes from two tables.

**Table 6.2.3: Querying two attributes in two tables (two databases)**

| Tables Queried | Data Source 1 Orcl | | | Data Source 2 Orcl2 | | |
|---|---|---|---|---|---|---|
| | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** |
| Genre (Adult) Language (French) | 7525.67 | 849.33 | 742 | 7407 | 854.333 | 722 |
| Language (Spanish) Country (USA) | 8522.67 | 755.33 | 1637 | 8510.33 | 631.333 | 1637 |
| Genre (Romance) Country (UK) | 10660 | 2765.5 | 1247 | 11410.3 | 859.25 | 1207 |
| Genre (Comedy) Language (French) | 1431.2 | 979 | 2746 | 1309.6 | 845.4 | 445 |
| Country (Mexico) Language (Spanish) | 1449.8 | 874.8 | 7135 | 1070.6 | 854.2 | 7135 |

It can be seen from the **Table 6.2.3**, that there is no particular relationship between the number of records fetched and the mapping and query times for the queries. The query times for both the data sources differ from each other by at least 20 milliseconds and almost by ~ 1000

milliseconds, for any number of records fetched in the same query for the two data sources. The mapping times for both the data sources are more or less the same except in the case of record 3.

**Table 6.2.4** shows the mapping times, query times and records fetched for querying with two attributes from two tables.

**Table 6.2.4: Querying single attribute with three attribute in a single table (two databases)**

| Tables Queried | Data Source 1 Orcl | | | Data Source 2 Orcl2 | | |
|---|---|---|---|---|---|---|
| | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** |
| Language (English, Hindi, Spanish) | 11641.25 | 990 | 6622 | 11359.8 | 1003.5 | 6622 |
| Country (USA, India, Mexico) | 1510.75 | 1144.25 | 7807 | 1702.25 | 1043 | 7807 |
| Language (Italian, French, German) | 994 | 921.67 | 32057 | 1149 | 865 | 32057 |
| Country (UK, France, Austria) | 1921.5 | 1793 | 36837 | 1160 | 926 | 36837 |
| Genre (Sport, Adult, Romance) | 5779 | 926 | 43466 | 5617 | 1329 | 42975 |
| Genre (Adult, Crime, Horror) | 5210.33 | 922 | 43559 | 5314 | 901 | 42774 |

**Table 6.2.5: Querying three attribute in three tables (two databases)**

| Tables Queried | Data Source 1 Orcl | | | Data Source 2 Orcl2 | | |
|---|---|---|---|---|---|---|
| | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** | **Mapping Time (msec)** | **Query Time (msec)** | **Records fetched** |
| Genre (Crime) Language (Italian) Country (Italy) | 2566.5 | 886.5 | 270 | 2683.5 | 800.75 | 250 |
| Genre (Romance) Language (English) Country (UK) | 8146.8 | 1171.3 | 1204 | 9862.6 | 1573.16 | 1138 |
| Genre (Sport) Country (USA) Language (English) | 5421.75 | 941.75 | 1622 | 5425.75 | 863 | 1577 |

It is evident from **Table 6.2.4** and **Table 6.2.5**, that there is no particular relation between the mapping times, query times with the records fetched for each query. The query times and mapping times for the different database in the same record differ by at least 80-90 milliseconds. This is primarily due to difference in the number for records in the two databases.

We observed that the mapping time and the query processing time did not show any particular trend with respect to the records fetched.

This could be attributed to several reasons, some of them mentioned in the Section 6.1.In this section, we use two databases, which have a different source of data i.e. data is different.

Processing the data in the two databases will take variable times. This is another reason for the variable query timings. The mapping times have increased for corresponding cases in section 6.1, since the concepts in the user ontology are now mapped to multiple (more specifically 2) ontologies.

## 6.3 Querying a Single Data Source Using Single User, Multiple Database Ontologies, with Missing Leaf Nodes

This case was implemented to evaluate the mapping technique adopted in this framework. We were also able to assess the functionality of the querying processing engine in constructing appropriate queries to question underlying data sources for missing or incomplete information in data sources.

In this particular case, the user Ontology had two sub categories, ‘Action 1’ and ‘Action 2’, for the genre ‘Action’ of movies. The data source 1 has categories of genre Action as ‘Action 1’ and ‘Action 2’, while data source 2 does not contain sub categories of the genre Action. Thus, the sub categories Action1 and Action2 are absent in the second data source. Since the user is able to view on the User Ontology, he/she queries the data sources for movies with genre Action1.

The system was tested to see if it was able to correctly map the appropriate concepts in the user ontology with concepts in the two ontologies. In this case, the query processing engine was expected to assume that the distribution of records in both the data sources is the same, and give a probability for the number of records in the other database. It has been found that the mapping techniques and query processing engine gave results with a high level of accuracy. Following are the results:

User Query: Select count from orcl.Movie Where genre = Action1
Results: 3183
Calculations for the above query:
Data Source1 (Database Name: orcl)
Total Number of Movies with Genre (Action) = 15062
Data Source2 (Database Name: orcl2)

Total Number of Movies with Genre (Action1) = 12664

Total Number of Movies with Genre (Action2) = 47272

Percentage of records with genre (Action1) in orcl2 = [12664 / (12664 + 47272)]

= 21.13 %

Percentage of records with genre (Action1) in orcl = (22 /100) * 15062

= 3182

*The difference between the observed calculated values us only 1 record.*

Summary of observations and calculations for queries are as follows:

**Table 6.3.1: Observed and calculated values for querying missing leaf nodes in multiple data sources**

| **Case** | **Concept queried** | **Available in Data Source 1** | **Available in Data Source 2** | **Observed Value** | **Calculated Value** | **Difference in Observed & Calculated value** |
|---|---|---|---|---|---|---|
| 1 | Action1 | Y | N | 3183 | 3182 | 1 |
| 2 | Action2 | Y | N | 11880 | 11879 | 1 |
| 3 | History1 | N | Y | 2054 | 2074 | 20 |
| 4 | History2 | N | Y | 3243 | 3264 | 21 |

It may be noted that the difference in the observed and the calculated value for case 3 and case 4 is large as compared to the difference in the observed and calculated values in Case 1 and Case 2.

There are some reasons which account for the difference in the observed and calculated values. It may be observed that the percentage of the number of records for the concept being queried (i.e. the missing attribute) to the number of records of its parent concept, is very small in cases 3 and 4.

Also the number of total records from data source 2 (i.e., the data source in which the concept is missing) for Case 3, Case 4 is small as compared to the other two cases, i.e. the concept which forms the parent concept for the concept being queried has fewer records as compared to Case 1 and Case 2.

Also, the mathematical calculations in the system are computed using the Java Math Library, which computes math functions differently. It has been found that the Math Library generates a different value of weighted percentage for the training data source (i.e., data source 1, which has the concept being queried), as compared to the computation of the weighted percentage using a simple calculator. There is a difference of 0.20 – 0.50 units of measurement, in each calculation depending on the total number of records fetched. This difference in the calculations increases with the decrease in the number of records fetched for the attribute.

Since the number of records is always a natural number, we take a ceil value for the weighted percentage applied to the records in the second data source. This also leads to a minor difference in the computation of the probabilistic number of records in data source 2.

## 6.4 Naïve Bayes Algorithm Applied to a Single User Ontology and a Single Database Ontology

This section is used primarily to acquire knowledge from the data stored in the relational data sources. We will test the Naïve Bayes Algorithm implemented for relational databases, as explained in the Section 2.5.

The following table summarizes the percent accuracy for the implementation of the Naïve Bayes Classifier tested on a variable number of records, using both the Independent Value formula and the Average Value formula.

**Table 6.4.1: Accuracy results for Naïve Bayes Algorithm on variable size test sets obtained from data source 1 (using Average value formula and the Independent value formula), data source 1 is also used for training.**

| | No of Samples tested | % Accuracy (Independent Value formula) | % Accuracy (Average Value formula) |
|---|---|---|---|
| **Training Dataset:** Database D1 | 10 | 10/10 (100 %) | 10/10 (100 %) |
| | 20 | 20/20 (100 %) | 20/20 (100 %) |
| | 30 | 30/30 (100 %) | 29/30 (96.66 %) |
| | 40 | 39/40 (97.5 %) | 38/40 (95 %) |
| **Testing Dataset:** Subsets of the Database D1 | 50 | 49/50 (98 %) | 48/50 (96 %) |
| | 60 | 57/60 (95 %) | 56/60 (93.33 %) |
| | 70 | 66/70 (94.28 %) | 65/70 (92.85 %) |
| | 80 | 76/80 (95 %) | 75/80 (93.75 %) |
| | 90 | 83/90 (92.22 %) | 82/90 (91.11 %) |
| | 100 | 91/100 (91 %) | 89/100 (89 %) |

From **Table 6.4.1**, it may be observed that the % accuracy obtained by implementing the independent value formula is higher than the % accuracy obtained by implementing the average value formula.

## 6.5 Naïve Bayes Algorithm Applied to a Single User Ontology, Multiple Database Ontologies

This section is primarily used to show, how knowledge acquired from data source 1 in the calculation of the Naïve Bayes algorithm is used to further predict knowledge in data source 2.

The following table shows the results for the Naïve Bayes implementation using data source 1 for training and data source 2 for testing.

**Table 6.5.1: Accuracy results for Naïve Bayes Algorithm on variable size test sets obtained from data source 1 (using Average value formula and the Independent value formula), data source 2 is used for training.**

| | No of Samples tested | % Accuracy (Independent Value formula) | % Accuracy (Average Value formula) |
|---|---|---|---|
| **Training Dataset:** Database D1 | 10 | 10/10 (100 %) | 10/10 (100 %) |
| | 20 | 19/20 (95 %) | 19/20 (95 %) |
| | 30 | 29/30 (96.66 %) | 28/30 (93.33 %) |
| | 40 | 38/40 (95 %) | 38/40 (95 %) |
| **Testing Dataset:** Database D2 | 50 | 47/50 (94 %) | 46/50 (92 %) |
| | 60 | 56/60 (93.33 %) | 55/60 (91.66 %) |
| | 70 | 64/70 (91.43 %) | 62/70 (88.57 %) |
| | 80 | 71/80 (88.75 %) | 69/80 (86.25 %) |
| | 90 | 78/90 (86.66 %) | 75/90 (83.33 %) |
| | 100 | 85/100 (85 %) | 81/100 (81 %) |

From **Table 6.5.1**, it may be observed that the percent accuracy obtained by implementing the Independent value formula is higher than the percent accuracy obtained by implementing the Average value formula.

From Section 6.4, 6.5, we can see that the independent value formula performs better than the average value formula. This could be because of the ability of the independent value formula to make use of multiple predictive values within a multiset, which makes it possible for the classifier to capture more multiset information.

# CHAPTER 7 - Conclusions

An ontology is a formal representation of the concepts in a domain and the relationships that exist between these concepts. Knowledge in various domains is stored in multiple distributed semantic heterogeneous databases. It is challenging to query specific data from these data sources. These databases can however be supported by ontologies describing them. This will lead to the semantic heterogeneity in the databases being reflected in the data source ontologies describing them. Thus, the semantic heterogeneity in the databases can be resolved by resolving the heterogeneity in the ontologies.

In this thesis, the problem of querying heterogeneous data sources has been addressed by constructing ontologies that describe the data sources. A user ontology, representing the user perspective of the domain, helps the user query the data sources. A mapping algorithm maps the user ontology to the data source ontologies, resolving the heterogeneity in the ontologies and therefore in the databases.

The problem of structural heterogeneity in the ontologies has been addressed by considering the ontology in the form of an XML tree, wherein nodes of the tree represent the concepts in a domain while the edges determine the relations between them. The lexical heterogeneity in the ontologies has been resolved using string based matching, lexical analysis and setting up heuristics to compare the attributes of the concepts, thereby determining the similarity between the concepts. It has been found that the above approaches used to resolve the heterogeneity in the database are effective in resolving the semantic and lexical heterogeneity to a high level of accuracy. The concepts in the ontologies were mapped appropriately, including missing leaf nodes. The mapping algorithm also mapped the subclass or super class relationships appropriately.

It has also been observed that the query processing engine was able to transform user queries into SQL inline queries that the Oracle database could comprehend. This was further confirmed by the result sets returned by the queries.

We have performed experiments to check the efficiency of the mapping algorithm by recording the time taken by the algorithm to map the user ontology concepts to the data source ontology concepts. We have found that the difference in time for mapping concepts, from the user and the data source ontologies, is not very large. The difference in the mapping times is

more pronounced with the increase in the number of concepts to be mapped. This is acceptable, considering that the increase in the mapping times does not exceeds 100 milliseconds.

The results of our experiments show that string based comparison and lexical analysis of the concepts are effective in mapping concepts in ontologies. The attribute based comparison, in which the attributes of each concept were compared on the basis of their domain, range and cardinality helped the mapping process in predicting the concept mappings to higher degree of accuracy. We can thus assert that we have achieved the goal of resolving heterogeneity in the data sources to a great degree of accuracy.

Experiments were also performed to evaluate the performance of the query processing engine by recording the time for translating the user query to a query, which the database understands, together with the time taken for the query to fetch the result set from the database. It was found that the query times, increased with increase in the number of concepts queried from the database. Since the databases under consideration are relational databases, the increase in the number of attributes being queried also increases the number of tables being queried. This also contributes to the increase in the query time, with the time taken by the query processing engine to formulate the database query.

Taking into consideration other factors affecting the query times, we can conclude that the query processing time for any query is not very high thereby making the system efficient in the querying of databases.

Experiments were also performed to check the performance of the system, to predict the weighted probability of the number of records for a concept in a data source, although the concept is not actually present in the data source. It was assumed that the distribution of records in the multiple data sources was the same. The system was able predict well the approximate number of records for the concepts missing in the data source.

The Naïve Bayes algorithm which was used to learn data from heterogeneous data sources was able to use the information in one data source (data source 1), to effectively predict the probabilities for the other data source (data source 2). The execution of the Naïve Bayes algorithm also tested the performance of the query processing engine and the mapping algorithm, since they are used to map the concepts and construct queries needed to compute the corresponding probabilities.

We can assert that we have been able to successfully design and develop a tool to query multiple heterogeneous data sources. The mapping algorithm implemented in the system to resolve heterogeneity, in congruence with the query processing engine performs as desired. We are able to map ontologies semi automatically, resolving the heterogeneity in the ontologies and inevitably in the databases.

The query processing engine performs efficiently by formulating the appropriate queries, in a reasonable amount of time. Although the query processing engine is primarily build to handle relational database, it will also work with non-relational databases, which identify SQL inline queries, e.g. MySQL.

This work has contributed to querying multiple heterogeneous data sources both relational and non-relational. Although several approaches have been suggested to resolve the semantic heterogeneity in multiple databases, most have taken into consideration non-relational databases. The few approaches which take into consideration non-relational databases use multi agents to resolve the problem. Also it has been observed that most approaches resolve either the lexical or the structural heterogeneity, but very few have resolved both and queried the databases to evaluate the mapping and the query processing techniques. We have incorporated a mapping technique which not only resolves lexical heterogeneity but also structural heterogeneity, thereby resolving heterogeneity to higher level of accuracy. We have further verified the functioning of the system, by using the Naïve Bayes algorithm to acquire knowledge from the database, which will be a real world application of this system. This system has been built with the purpose of being used for learning classifiers for multiple heterogeneous data sources. It should be possible to design other learning classifiers, to acquire knowledge from multiple heterogeneous data sources.

# CHAPTER 8 - Future Work

This chapter briefly discusses the areas of related work, which we may want to address in the near future in our system. In doing so, we will discuss challenges and opportunities ahead in the field of querying heterogeneous data sources.

We have presented an approach to ontology mapping by resolving lexical and structural heterogeneity in the data source ontologies based on modeling and linguistic analysis. The purpose of the work is to introduce a method for finding semantic correspondence among the ontologies with the intention to support interoperability of information systems.

The current situation with heterogeneous data sources in distributed modeling suggests a need for adjustable variables in the mapping algorithm depending on the application domain, the modeling group and the organization. This way the ontology mapping or integration tool will be able to support a multi lingual (OWL Lite, OWL DL, OWL Full, RDF), multi domain heterogeneous data source.

The following are problems related to ontology mapping, which could be addressed in future work.

## 8.1 Mapping Ontologies with Missing Concepts

The experiments presented in this thesis are based on the assumption that the local (user) ontology is 'complete'. However if the local (user) ontology is not 'complete', then it would be almost *impossible* to map concepts in the user (local) ontology to a data source ontology.

However, it is possible that local ontology does not contain a particular concept in the domain. An example would be that the local ontology for the domain of Movies may not include the concept of 'Certification" which is a certificate obtained to release in a particular country, but the data source ontology has a specific mention of the concept Certifications. Since the user (local) ontology represents a user's perspective of the domain of Movies, the user will never know about the existence of the concept, due to its absence in the user (local) ontology.

However, if it were possible to utilize the user and database ontologies to further compile a consolidated local (user) ontology, better describing the domain, it would then be possible to map missing concepts in the user ontology. Thus we would need to use the initial set of

ontologies to integrate into another ontology, which can better represent the domain, combining all the concepts from the multiple Ontologies.

## 8.2 Mapping Ontologies in Multiple Ontology Languages

The experiments performed in this thesis work only with the OWL Lite, Web Ontology Language. It may be realized that not all ontologies on the semantic web are available in this particular language. Thus, aligning two ontologies expressed in different languages, depending on their degree of expressivity can be done based on the mapping of different logical languages.

At present it is possible to perform translations between languages at the ontology schema level. These translations are purely syntactic and are expressed using transformation languages like XSLT. Thus, we can attempt to solve the problem of integrating different modeling paradigms having different semantics.

## 8.3 Integrating the Two Versions of the Ontology

At present, we have multiple heterogeneous data sources corresponding to multiple data source ontologies. It would be interesting to integrate the multiple ontologies into a single data source ontology describing the data in all the data sources. We may then map the user (local) ontology to the single homogeneous data source ontology. This data source ontology could then be used to query the heterogeneous data sources, which could hopefully lead to results similar to those obtained by using different data source ontologies.

## 8.4 Usage of Different Mapping Techniques

There are several mapping techniques to map lexical and structural heterogeneity between two ontologies. We could add an array of different mapping techniques to compare the mapping technique used by us in the present application.

## 8.5 Functional Mappings

There are certain complex mappings in ontologies which may require the use of aggregate function to express the differences in the ontology models. Examples of such functions would include functions for currency conversions, unit – measurement conversion from one ontology to another ontology. This could also include string, integer manipulation functions. It would be interesting to have a mapping language, which could express such functions when mapping two ontologies using different unit representations to dynamically get the necessary transformation. We are already implementing a functional approach to resolve the problem of currency conversion. However, we need to design a standard technique, which can be implemented in any mapping architecture.

## 8.6 Consolidated Results from Data Sources

The system could be designed to return a consolidated result set, by combining the results it obtains from each of the data sources. We could thus remove any redundancy in the records returned to the user.

## 8.7 Inference Engine for Resolving Structural Heterogeneity

In the present state, we assume the existence of two inference rules for resolving the structural heterogeneity in the ontology graphs, providing formal representation for the data sources. However, it should also be possible to include an inference engine, which studies the structure of the two ontology graphs and infers different structural similarity rules for different ontologies.

## 8.8 Different Types of Queries

At present, the system is limited to formulating SQL inline queries. We could broaden the array of queries that can be formulated in this system, thereby improving the functionality of the system.

## 8.9 Representative Algorithms for Learning Classifiers from Distributed Data

In the present implementation, we have already implemented the Naïve Bayes algorithm for heterogeneous data sources [21]. However, it is necessary to design and implement also other algorithms for learning from distributed data. Examples of such algorithms include Nearest Neighbor algorithm, Bayes Network and Decision Tree classifiers. This will also help us better understand about the representation of data in the multiple heterogeneous data sources. Also this will help us establish the precise conditions under which the proposed algorithms offer significant savings in bandwidth, memory, and/or computation time (relative to their centralized counterparts) [12].

## 8.10 Different Types of Data Sources

The system is designed for relational data sources. Although the query processing engine is designed such that it constructs queries for relational data sources, it may be noticed that these queries can also be applied to non-relational databases, which identify SQL inline queries. We could further extend the range of data sources to flat files, XML based data sources, object oriented data sources etc.

# References


1. Semantic Web Vision: survey of ontology mapping systems and evaluation of progress, Arshad Saleem (2006). Masters Thesis, School of Engineering, Blekinge Institute of Technology.
2. Translation approach to portable ontology specification, T. R. Gruber (1993). Knowledge Acquisition, Vol. 5, No. 2.], pp. 199-220.
3. Ontology mapping: the state of the Art, Yannis Kalfoglou and Marco Schorlemmer (2003). The Knowledge Engineering Review, Volume 18, Issue 1.
4. Automatic Ontology Mapping for Agent Communication, F. Wiesman, N. Roos, P. Vogt (2002). MERIT-Infonomics Research Memorandum series. http://www.infonomics.nl.
5. OWL: Representing Information Using the Web Ontology Language by Lee W Lacy, (2005).
6. Agents and the Semantic Web. IEEE Intelligent Systems, James Handler (March/April 2001). Intelligent Systems, IEEE Volume 16, Issue 2, Mar-Apr 2001 Page(s): 30 - 37
7. Semantic Enrichment for Ontology Mapping, Xiaomeng Su, Ph.D. Thesis (2004).
8. Ontology mapping through analysis of model extension, Xiaomeng Su, Terje Brasethvik and Sari Hakkarainen (2003).pp. 101-104, Short Paper Proceedings of the 15th Conference on Advanced Information Systems Engineering,, CEUR Workshop Proceedings, Technical University of Aachen (RWTH), Klagenfurt, Austria, Vol. 74, June 2003.
9. Dynamic Aspects and Semantic Enrichment in Schema Comparison, S Hakkarainen. PhD thesis, Stockholm University, 1999.
10. Ontology-Driven Information Extraction and Knowledge Acquisition from Heterogeneous, Distributed, Autonomous Biological Data Sources, Adrian Silvescu, Jaime Reinoso-Castillo, and Vasant Honavar, et al. – 2001.in: Proceedings of the IJCAI-2001 Workshop on Knowledge Discovery from Heterogeneous, Distributed, Autonomous, Dynamic Data and Knowledge Sources, 2001.
11. Querying Heterogeneous Information Sources Using Source Descriptions, A.Y.Levy, A. Rajaraman and J.J. Ordille (1996).Proceedings of the 22th International Conference on Very Large Data Bases, Pages: 251 – 262, ISBN: 1-55860-382-4.

12. Learning classifiers from Distributed, Semantically Heterogeneous, Autonomous Data Sources, Caragea, D. (2004), Ph.D. Thesis. Department of Computer Science, Iowa State University, Ames, Iowa, USA
13. Query Translation for Ontology-extended Data Sources, Jie Bao, Doina Caragea, Vasant Honavar (2007).In: Proceedings of the AAAI 2007 Workshop on Semantic e-Science, Vancouver, Canada.
14. C-owl: Contextualizing Ontologies, Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H. (2003).In: Second International Semantic Web Conference (ISWC-2003), LNCS vol. 2870, pp. 164-179, Springer Verlag, 2003.
15. Algorithms and Software for Collaborative Discovery from Autonomous, Semantically Heterogeneous, Distributed Information Sources, Caragea, D., Zhang, J., Bao, J., Pathak, J., and Honavar, V. (2005). In: Proceedings of the 16th International Conference on Algorithmic Learning Theory. Lecture Notes in Computer Science. Singapore. Vol. 3734, Pp. 13-44. Berlin: Springer-Verlag.
16. http://wordnet.princeton.edu/
17. Learning From Attribute Value Taxonomies and Partially Specified Instances, Zhang, J. and Honavar, V. (2003). In: Proceedings of the International. Conference on Machine *Learning* (ICML 2003), pp: 880-887.
18. Knowledge Discovery in Multiple Databases, Zhang, S., Zhang, S., and Wu, X. (2004).
19. Machine Learning by Tom M. Mitchell, McGraw Hill (1997)
20. Simple Estimators for Relational Bayesian Classifiers, Jennifer Neville, David Jensen and Brian Gallagher (2003).In: Proceedings of the Third IEEE International Conference on Data Mining, Page: 609.
21. Learning Relational Bayesian Classifiers on the Semantic Web, Doina Caragea, Jie Bao and Vasant Honavar (2007).In: Proceedings of the IJCAI 2007 Workshop on Semantic Web for Collaborative Knowledge Acquisition (SWeCKa 2007). In conjunction with the Twentieth International Joint Conference on Artificial Intelligence, Hyderabad, India, January 2007.
22. Ontology based Query Processing in Database Management Systems, Chokri Ben Necib and Johann-Christoph Freytag (1999). Journal title: Lecture Notes in Computer Science, Bibliographic details 2003, ISSU 2888, pages 839-857 Publisher: Springer-Verlag; 1999

23. A Environment for merging and testing large ontologies, Deborah L. McGuinness, Richard Fikes, James Rice, Steve Wilder(2000). In Cohn, A., Giunchiglia, F., Selman, B. (eds.), and KR2000: Principles of Knowledge Representation and Reasoning, San Francisco (2000) 483-493
24. Querying Heterogeneous and Distributed Data Repositories using Ontologies by Eduardo Mena, Arantza Illarramendi (1997). In Proceedings of the 7th European-Japanese Conference on Information Modeling and Knowledge Bases (IMKB'97)
25. OntoShare: Using Ontologies for Knowledge Sharing, John Davies, Alistair Duke and Audrius Stonkus (2002). In Proceedings of the WWW2002 Semantic Web workshop, 11th International WWW Conference WWW2002, Hawaii, USA, 2002.
26. Semi-automatic semantic discovery of properties from database schemes L. Palopoli, D. Sacca and D. Ursino (1998). In Proceedings IDEAS'98, 244-253, Cardiff, UK, 1998.
27. A multi-agent system for querying heterogeneous data sources with ontologies, Paolo Dongilli, Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris (2005).In: Proc. of the 13th Italian Sym. on Advanced Database Systems (SEBD 2005).
28. Si-designer: A tool for intelligent integration of information, D. Beneventano, S. Bergamaschi, I. Benetti, A. Corni, F. Guerra, and G. Malvezzi(2001).International Conference on System Sciences (HICSS2001)
29. The MOMIS approach to information integration, D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini (2001). In AAAI International Conference on Enterprise Information Systems (ICEIS 2001).
30. A data integration framework for e-commerce product classification, S. Bergamaschi, F. Guerra, and M. Vincini (2002). In: Proceedings of the First International Semantic Web Conference on The Semantic Web, 2002
31. Semantic integration of semistructured and structured data sources, S. Bergamaschi, S. Castano, and M. Vincini. (1999). *SIGMOD Record,* 28(1):54{59, March 1999
32. Combining Heterogeneous Data Sources through Query Correspondence Assertions, Ulf Leser (1998). Workshop on Web Information and Data Management 1998: 29-32
33. Query Processing for heterogeneous Data integration using Ontologies, Huiyong Xiao: PHD Thesis (2006), University of Illinois at Chicago, IL, USA.

34. Evaluating ontology-mapping tools: Requirements and experience, N. F. Noy and M. A. Musen (2002). In: Presented at 13th International Conference on Knowledge Engineering and Knowledge Management, Siguenza, Spain,
35. Ontology of integration and integration of ontologies, D. Calvanese, D. G. Giuseppe, and M. Lenzerini (2001).In *Description Logic Workshop (DL 2001)*, pages 10–19, 2001.
36. *Automatic Ontology Mapping for Agent Communication,* F. Wiesman, N. Roos, P. Vogt. In Proceedings of BNAIC '01, pages 291 - 298. BNAIC, 2001.
37. Agent Communication Languages: Rethinking the Principles, Munindar P. Singh (1998). In Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, Honolulu, Hawaii.
38. Learning to Map between Ontologies on the Semantic Web**,** AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy (2002).In *Proceedings of the World-Wide Web Conference (WWW-2002)*, 2002.
39. *A Survey of Recent Research in Ontology Mapping,* Davis Marques. Surrey, BC Canada
40. Ontology Mapping using Background Knowledge, Zharko Aleksovski, Michel Klein.KCAP'05, October 2–5, 2005, Banff, Alberta, Canada.
41. T*he problem of ontology alignment on the web: a first report*, Davide Fossati, Gabriele Ghidoni, Barbara Di Eugenio, Isabel Cruz, Huiyong Xiao, Rajen Subba (2006).The 2nd Web as Corpus Workshop (associated with the 11th Conference of the European Chapter of the ACL), April, 2006.
42. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment, Natalya Fridman Noy and Mark A. Musen (2000).In:Proceedings 17th Intl. Conf. on Artificial Intelligence (AAAI 2000)
43. http://www.w3.org/TR/2004/REC-owl-guide-20040210/#Introduction
44. Comparison of personal ontologies represented through conceptual graphs, R. Dieng and S. Hug (1998). In *Proceedings 13th ECAI*, Brighton (UK), pages 341–345, 1998.
45. State of the Art on Ontology Alignment, J. Euzenat, T. L. Bach, J. Barrasa, P. Bouquet, J. D. Bo (2004). Knowledge Web, Statistical Research Division, Room 3000-4, Bureau of the Census, Washington, DC, 20233-9100 USA, deliverable 2.2.3, 2004.
46. Processing XML with Java, Elliotte Rusty Harold (2001).

47. JDOM and XML Parsing, Part 1, *Jason Hunter,* (2003), [*www.**jdom**.org/docs/oracle/**jdom-part1**.pdf]*
48. JDOM and XML Parsing, Part 2, Jason Hunter, (2003), [www.**jdom**.org/docs/oracle/**jdom-part1**.pdf]
49. JDOM in the Real World, Part 3, Jason Hunter, (2003), [www.**jdom**.org/docs/oracle/**jdom-**part3.pdf]
50. MIT Java WordNet Interface: User's Guide, Mark A. Finlayson (2008).
51. Oracle 10*g* Grid & Real Application Clusters, Oracle10*g* Grid Computing with RAC, *Mike Ault, Madhu Tumma* (2004).
52. http://protege.stanford.edu/
53. A comparative analysis of methodologies for database schema integration, C. Batini and M. Lenzerini (1996). ACM Computing Surveys, 18, 4, December 1986, pp. 323-364.
54. Ontology Mapping Survey, Siyamed Seyhmus Sinir (2005).In Proceedings of IJCAI 1999.
55. An ontology-extended relational algebra, Bonatti, P., Deng, Y., and Subrahmanian, V. (2003). In: Proceedings of the IEEE Conference on Information Integration and Reuse, IEEE Press (2003) 192–199
56. Learning Probabilistic Relational Models. In: Relational Data Mining, Getoor, L., Friedman, N., Koller, D. and Pfeffer, A. (2001).
57. Learning Classifiers from Semantically Heterogeneous Data, Caragea, D., Pathak, J., and Honavar, V (2004). In: Proceedings of the Third International Conference on Ontologies, Databases and Applications of Semantics for Large Scale Information Systems (ODBASE 2004), Springer-Verlag, Lecture Notes in Computer Science. October 25-29, 2004, Agia Napa, Cyprus. Vol. 3291, Pp.963-980. Springer-Verlag.
58. Towards the Semantic Web: Ontology-driven Knowledge Management, Dr John Davies, Professor Dieter Fensel and Professor Frank van Harmelen (2003).ISBN-13, 9780470848678.Förlag, JOHN WILEY & SONS LTD.
59. Distributed description logics: Directed domain correspondences in federated information sources, Alexander Borgida and Luciano Serafini.Lecture Notes in Computer Science; On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002

60. A Language to Specify Mappings between Ontologies, Francois Scharffe, Jos de Bruijn (2005). *IEEE Conference on Internet-Based Systems (SITIS6)*, December 2005, Yaounde, Cameroon.