A MULTIAGENT ARCHITECTURE FOR SEMANTIC QUERY ACCESS TO LEGACY RELATIONAL DATABASES

by

Mohammad Zubayer

B.Sc., International Islamic University Malaysia, 2005

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
MATHEMATICAL, COMPUTER, AND PHYSICAL SCIENCES
(COMPUTER SCIENCE)

UNIVERSITY OF NORTHERN BRITISH COLUMBIA
November 2011

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-87590-2

Our file Notre référence
ISBN: 978-0-494-87590-2

Abstract

This thesis proposes a novel approach to accessing information stored in legacy relational databases (RDB), based on Semantic Web and multiagent systems technologies. It introduces an architectural model of the Semantic Report Generation System (SRGS), designed to address the rising demand for flexible access to information in decision support systems. SRGS is composed of server Database Subsystems (DBS) and client User Subsystems (US). In a DBS, an agent interacts with the administrator to build a reference ontology from the RDB schema, which enables semantic queries without modifying the database. In a US, the decision-making user accesses the system through a simplified natural language interface, using customized extensions to the reference ontology that was imported from DBS; an agent helps build the custom ontology, and facilitates query formulation and report generation. The proposed approach is illustrated by several scenarios that highlight the key behavioral aspects of accessing information and developing ontologies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The importance and impact of computer-based information systems in the progress of human society is well acknowledged in all disciplines. These systems enable users to create, store, organize, and access large volumes of information. Individuals and organizations increasingly rely on them for problem solving, decision making, and forecasting. The requests for information are increasing in complexity and sophistication, while the time to produce the results is tightening. These trends in using information systems compel researchers to look for more effective access techniques that meet modern requirements.

Computer-based systems rely on databases for storing information. The relational database model (Codd, 1970) has been dominant for more than three decades. In order to extract the necessary information from relational databases (RDB), non-technical users require technical assistance of database programmers, report writers, and application software developers. These support tasks may be time consuming and involve multiple technical experts, resulting in delays and costs. In order to speed up access and give users more control, decision support systems rely on data warehousing techniques.

Those techniques require information to be extracted from operational databases, reorganized in terms of facts and dimensions, and stored in data warehouses (Olszak and Ziemba, 2007). Operational databases are designed to support typical day-to-day operations, whereas data warehouses are designed for analytical processing of large volumes of information accumulated over time. That approach still relies on human technical expertise and may require weeks to effect the restructuring of data. Moreover, it requires accurate foresight as to what information might be needed.

In the meantime, two relevant technologies have developed in the realm of the World Wide Web and Artificial Intelligence. One is the Semantic Web, which is a web of data that enables computers to understand the semantics, or meaning, of information on the Web (Berners-Lee, 1998a). The development of the Semantic Web entails structuring of information using a set of tools and standards recommended by the World Wide Web Consortium (W3C). This process requires formal representation of human knowledge in the form of a hierarchy of ontologies that correspond to knowledge domains at different levels of abstraction. An ontology is an explicit specification of a conceptualization; a conceptualization is an abstract, simplified view of the world that is represented for some purpose (Gruber, 1993). The Semantic Web infrastructure is in an early stage of construction; it is developing through numerous current projects.

The other novel technology is based on intelligent software agents and multiagent software systems. A software agent is a computer program, which is situated in a specific environment, and can act autonomously in that environment in order to meet its delegated objectives (Wooldridge, 2009). Multiple interacting agents can form a single

system, known as a multiagent system (MAS). Agent technology brings together research results from the last few decades in several disciplines, mainly artificial intelligence, distributed systems, software engineering, economics, psychology, and social sciences. Following decades of research, this is becoming a major trend in mainstream computing and a likely successor to the currently dominant object-oriented software engineering paradigm (Lind, 2000).

Software agents are expected to assist humans in many tasks, including searching and reasoning with information in a decision support environment. However, the information underlying the decision process needs to be organized following Semantic Web structures such as ontologies. Software agents can reason with information in a knowledge base once it is organized using ontologies. In reasoning with information stored in a database, software agents operate on the Closed World Assumption (CWA), which states that the information in the database is complete, and what is not asserted as true, is false (Russell and Norvig, 2003).

In this thesis, I explore how a combination of these two technologies can be applied to overcome some of the issues arising in the context of traditional decision support environments. I propose a system architecture, Semantic Report Generation System (SRGS), that relies on Semantic Web tools and software agents to enable effective user access to information in RDB systems without depending on report writers and database programmers. In SRGS, direct access to information is achieved by building a layer of semantic information structures on top of the existing legacy RDB system, and allowing users to interact with the system using a Simplified Natural Language (SNL). SRGS employs a

software agent to help the Database Administrator in building ontologies using the structure of information stored in the RDB system, human domain knowledge, and knowledge resources on the Semantic Web. SRGS employs another software agent to help the users create their own layers of ontology by defining user-specific concepts that may not exist in the reference ontology developed from the RDB. This agent also assists the users in formulating requests for information.

My research started with the formulation of system requirements followed by an analysis leading to a preliminary definition of the system architecture, and proceeded to selective modeling of existing Semantic Web and MAS tools that fit into the architecture of SRGS. I carried out these two tasks in an iterative manner, in which I identified existing software components that could be integrated, and then refined the architectural definition so that it could rely on the identified components.

SRGS consists of two subsystems: the Database Subsystem (DBS) and the User Subsystem (US). The DBS facilitates creating, storing, and organizing information while the US allows accessing this information. The US and DBS can reside on different machines and communicate through a network. Within each subsystem there is a software agent that assists the human users in organizing and accessing information. I show three possible configurations of SRGS with regards to the multiplicity of the subsystems: multiple USs to single DBS, single US to multiple DBSs, and multiple USs to multiple DBSs. I also show how the multiplicity of users affects the architecture of SRGS. For instance, when multiple users access the US, some elements in the system are customized to suit each user's preferences for interacting with the system.

In the Database Subsystem (DBS), a software agent named Database Interface Agent (DBIA) assists a Database Administrator (DBA) in gradually building a *reference* ontology from the RDB structure; this is the common core ontology for all users of SRGS. The software agent possesses the technical know-how of ontology development process. In addition, the software agent refers to external resources on the Web such as publicly available libraries of ontologies and online lexical dictionaries, as sources of conceptual, lexical, and domain-specific knowledge. Without the Semantic Web in place, the software agent is limited to its technical knowledge of the ontology development process alone. Thus, the software agent's role evolves from being a technical assistant to a knowledgeable partner in the ontology development process as more domain-specific ontological resources become available with the development of the Semantic Web. The DBS is also responsible for retrieving the necessary information from the RDB system and presenting this information to the US in response to the request.

The User Subsystem (US) accepts a user's request for information, asks the DBS to retrieve the information, and presents it as a formatted report. The user develops a *custom* ontology, which complements the reference ontology by defining user-specific terms with the assistance of a software agent called User Interface Agent (UIA). The user-system interaction occurs in a simplified natural language. The user formulates requests for information and introduces new terms in the custom ontology using the simplified natural language. The ontologies are used to formulate and verify requests for information, construct semantic queries, and format the extracted information as reports.

The feasibility of the proposed approach is then substantiated using three scenarios illustrating the behavioral aspects of SRGS in accessing information in an RDB system and developing ontologies from the RDB structure. The scenarios show the interactions that occur between the agents and the human actors, the actions performed by the agents, and the tasks executed by the system components. In order to help to develop ontologies, the agents are equipped with meta-ontological knowledge and the know-how of methodological steps for guiding the ontology construction process. In addition, agents refer to external knowledge resources on the Semantic Web. The agents provide the technical knowledge while the human actors make decisions. The ontology construction process begins in the Database Subsystem with a rudimentary version of reference ontology generated through automatic conversion of an RDB structure to a Semantic Web structure. The converted structure, called the base ontology, then serves as a starting point from which the DBIA, in interaction with the DBA, incrementally develops a full reference ontology.

The first scenario illustrates how the user of SRGS can request for information using the SNL. It illustrates the specific tasks performed by each system component in formulating a request for information, retrieving the requested information from the underlying database, and presenting it to the user as formatted report. The second scenario is aimed at showing the interactions between the DBIA and the DBA, and the activities that occur within the DBS in the process of constructing the reference ontology. Once the construction completes, the DBS exports a copy of the reference ontology to the attached US. The third scenario focuses on showing how the user can complement the reference ontology by defining user-specific concepts in a custom ontology.

The remaining chapters of the thesis cover the background and related work (Chapter 2), the approach for query access to legacy RDB systems using Semantic Web tools (Chapter 3), the architectural model (Chapter 4), the behavioral aspects of SRGS (Chapter 5), analysis and evaluation (Chapter 6), and conclusions and future work (Chapter 7).

Chapter 2

Background and Related Work

This chapter presents the background and an overview of previous research work in Relational Database systems (Section 2.1), the Semantic Web (Section 2.2), developing ontologies from relational structures (Section 2.3), and multiagent systems (Section 2.4).

2.1 Relational database systems

The wealth of data that populates the Web is stored in legacy information systems; they are socio-technical computer-based systems that were developed in the past using older or obsolete technology. It may be risky to replace a legacy system, because an organization and its organizational policies can be critically dependent on its structure and function. Legacy information systems contain immense volumes of data accumulated over the lifetime of the system (Sommerville, 2004). Common sources of legacy data include relational, hierarchical, network, and object databases; as well as XML documents and flat files, such as the comma-delimited text files (Ambler, 2003). My study focuses on accessing information in relational databases, because of their prevalence in present

legacy information systems.

A relational database (RDB) is implemented using the relational data model invented by Codd (1970). It is based on the mathematical term *relation*, which is represented as *table* in the database context. Each relation (table) represents an entity and is made up of named *attributes* (columns) of that entity, and each row contains one *value* per attribute (Connolly and Begg, 2001). The relational model has been the dominant data modeling technique because of its track record of scalability, reliability, efficient storage, and optimized query execution (Sahoo et al., 2009). However, one major limitation of the relational model is its inability to capture semantic relationships between data units. In addition, non-technical users of relational databases require technical assistance of database programmers and database administrators in order to access the necessary information.

The remainder of this section is organized as follows: Knowledge representation using the relational mode is discussed in Subsection 2.1.1, and a language for querying information in RDB system is presented in 2.1.2.

2.1.1 Knowledge representation using RDB model

In relational data modeling, an entity is represented as a table, and each attribute of the entity becomes a column in that table. Each row is an instance of the entity and can be uniquely identified by a primary key. Relationships between entities are represented by foreign keys. This logical structure of a database is called the *database schema* (Connolly and Begg, 2001). Table 2.1 shows a subset of an RDB containing two tables: **Depart-**

ment and **Instructor**. Each department is uniquely identified by Department_Name and each instructor by Instructor_ID. An instructor belongs to only one department and a department can have one to many instructors. Therefore, Department_Name column is the primary key in the **Department** table and foreign key in the **Instructor** table. This model is for illustrative purpose only, and does not represent any real database.

Table 2.1: Sample RDB tables

Table: Department		
Department_Name	Building	Budget
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Finance	Painter	120000

Table: Instructor			
Instructor_ID	Last_Name	Salary	Department_Name
12121	Wu	90000	Finance
45565	Katz	75000	Comp. Sci.

2.1.2 The SQL query language

Structured Query Language (SQL) is the standard language for defining and manipulating data stored in RDB systems (Chamberlin and Boyce, 1974; IBM, 2006). Common SQL commands include schema creation and modification, data insert, query, update, and delete. Writing SQL queries requires understanding of the underlying database

schema, in addition to the knowledge of the SQL itself. Figure 2.1 shows a sample SQL query that returns all department names and budgets from the table Department.

```
SELECT Department_Name, Budget
FROM Department;
```

Figure 2.1: An SQL query

2.2 The Semantic Web

The Semantic Web is a web of data that enables computer systems to understand the semantics, or meaning, of information that populates the Web (Berners-Lee, 1998a). This is in contrast to the current Web which is a web of documents. The objective of the Semantic Web is driving the evolution of the current web of document into a web of data in which users can easily find, share and combine information. The Semantic Web will enable machines to understand the meaning of information, thus allowing machines to assist human users in finding right information. Currently there are many individual projects underway towards developing Semantic Web infrastructure and applications using common formats and technologies recommended by the World Wide Web Consortium (W3C) (Baker et al., 2009). Many of these applications are intended to eventually connect with each other and share information between them.

The information underlying the Semantic Web must be organized according to the meaning of the represented contents. Human knowledge can be represented by organizing information as *ontologies*. Ontologies are considered as one of the essential parts of the

Semantic Web. A university ontology, for instance, would define concepts such as faculty, student, department, course, project, etc., and how they are related to each other. At the most basic level of an ontology, concepts are represented as *classes*, and various attributes of a concept are represented as *properties*, and a class can have *subclasses* representing concepts that are more specific than the parent class. For example, a *student* class may have two subclasses: *undergraduate student* and *graduate student*. In addition to classification, one can define relationships between classes in an ontology. Thus, ontologies provide the structural framework for organizing and reasoning with information within a particular domain. In addition, *upper ontologies*, which describe general concepts that are the same across all knowledge domains, provide the functionality of semantic interoperability between multiple domain ontologies (Noy and McGuinness, 2001).

In the rest of this section, I briefly review the main technical concepts underlying the Semantic Web architecture as envisioned in the W3C standards. Subsection 2.2.1 provides the basic definitions. Subsection 2.2.2 describes the knowledge representation model, and 2.2.3 presents a corresponding query language for retrieving information. Finally, 2.2.4 outlines three early Semantic Web projects.

2.2.1 Semantic Web technologies

The development of the Semantic Web entails restructuring of information using a set of languages and standards. The Semantic Web architecture is illustrated in Figure 2.2. The main components in the Semantic Web stack are Unified Resource Identifier (URI), Resource Description Framework (RDF), RDF Schema (RDFS), Simple Protocol and RDF Query Language (SPARQL), and Web Ontology Language (OWL).

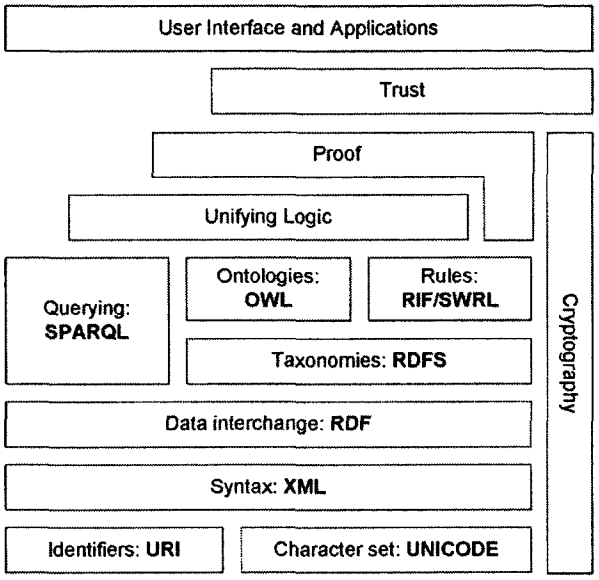

Figure 2.2: Semantic Web stack layer
(Reproduced from Wikipedia Semantic Web entry)

URI provides a mechanism for uniquely identifying each information resource on the Web. A special type of URI is Unified Resource Locator (URL) which uniquely identifies the location of a resource, such as a web page, within the World Wide Web (Sauermann et al., 2008). Internationalized Resource Identifier (IRI) is a generalization of URI that may contain characters from the Universal Character Set, including Chinese, Japanese and Korean. RDF is a data modeling language which conceptualizes a data unit as a resource in terms of its property and property-values. Each resource is uniquely identified by its URI (Manola and Miller, 2004). RDFS provides the basic primitives such as classes and properties for structuring RDF resources. SPARQL is a language for querying RDF data, analogous to the way SQL is used for querying relational data (Prud'hommeaux and Seaborne, 2007). OWL is a knowledge representation language for authoring on-

tologies. It also facilitates reasoning and inference. OWL is based on RDF and RDFS (McGuinness and Harmelen, 2004).

Discussing in details all the components of the Semantic Web is beyond the scope of this chapter. I focus on the components that are relevant to my research objective with regards to representing RDB structures in Semantic Web structures. In the following subsection, I illustrate how some of these components can be used to represent and access information in the Semantic Web compliant format.

2.2.2 Knowledge representation using the RDF model

RDF is based on the idea of describing an entity in terms of properties and property-values. An RDF statement consists of an entity (*the subject*), a property (*the predicate*) and a property-value (*the object*). This **subject-predicate-object** expression, also written as **(S, P, O)** is known as a *triple.* The complete description of an entity would consists of a collection of triples called an *RDF graph.* The entity's class, which is the table name in the relational model, is described by an RDF triple containing the **rdf:type** predicate. rdf:type is used to state that a resource is an instance of a class. For example, a triple of the form: **R rdf:type C** states that **R** is an instance of **C**, and **C** is an instance of **rdfs:Class** (Brickley and Guha, 2004). RDF mandates that each subject and predicate must be URIs; the object can be a URI or an actual value.

Relational databases can be converted into RDF triples by following the core guidelines outlined by Berners-Lee (1998b). He proposed the following direct mappings between RDB and RDF:

- An RDB record (row) is an RDF is an RDF subject
- The column name of an RDB is an RDF predicate
- An RDB table cell is an RDF object

Figure 2.3 shows RDF representation of an instructor's record from the previous example of the relational model. The first row from **Instructor** table can be written as "**45565** has **Last_Name** which is **Katz**". This statement becomes an RDF triple when written in the form **(S, P, 0)**, in other words, **(45565, Last_Name, Katz)**. But **S** and **P** must be in the URI format hence, the URI

`http://localhost:8080/resource/Instructor/45565` is assigned to **45565**, and

`http://localhostvocab/resource/Instructor_Last_Name` to **Last_Name**.

Therefore, the correct triple is

(`http://localhost:8080/resource/Instructor/45565`,
`http://localhostvocab/resource/Instructor_Last_Name`, "Katz")

In Figure 2.3, each triple corresponds to a single arc with its beginning node as the subject, arc label as the predicate, and ending node as the object.

RDF refers to a set of URIs as a *vocabulary* (Manola and Miller, 2004). An organization may define its own vocabulary consisting of the terms it uses in its business. In our example, such terms can be

`http://localhostvocab/resource/Instructor` for **Instructor**, and

`http://localhostvocab/resource/Instructor_Last_Name` for **Last_Name.**

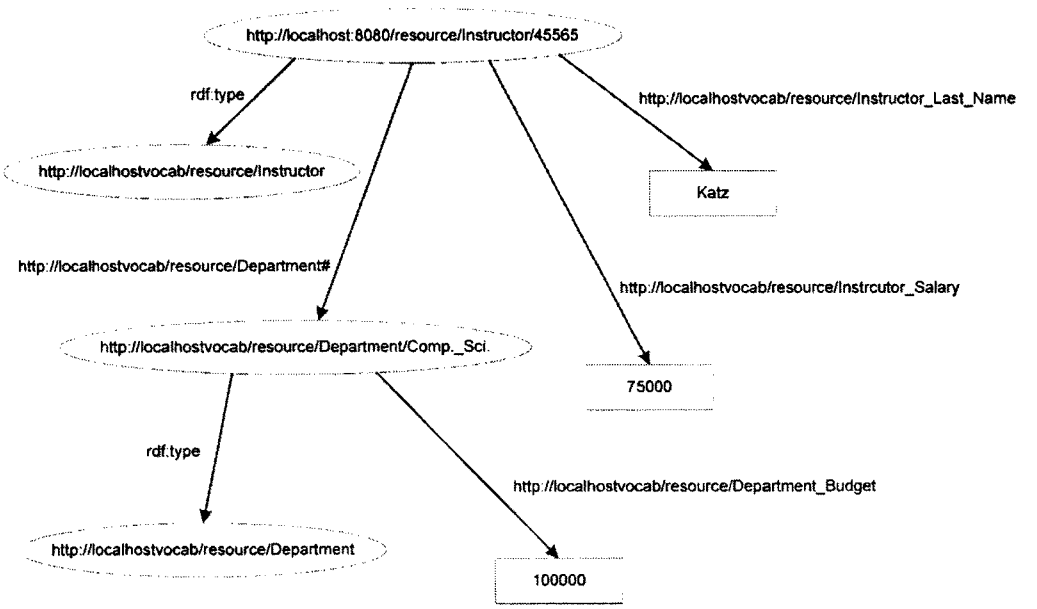

Figure 2.3: RDF graph showing an instructor's record

An organization might as well take advantage of an external vocabulary instead of defining its own. Friend of a Friend's (FOAF) Vocabulary Specification (Brickley and Miller, 2005) and Dublin Core's Metadata Terms (Powell et al., 2007) are examples of such vocabularies. In our example of the RDF model, one can use FOAF's

```
http://xmlns.com/foaf/spec/#term_lastName
```

instead of using our own

```
http://localhostvocab/resource/Instructor_Last_Name
```

to refer to the term **Last_name**. Constructing RDF statements with URI predicates instead of character strings offers two main benefits. First, it minimizes the practice of using different terms to refer to the same thing. For instance, a database designer may use attribute names such as *Family Name* or *Second Name* to refer to someone's last name.

In order to avoid the use of multiple attribute names, FOAF's Vocabulary Specification provides a unique identifier – `http://xmlns.com/foaf/spec/#term_lastName` – to refer to a person's last name. This mechanism forces a designer to use the same URI predicate for all occurrences of a last name. Second, the use of URIs in RDF triples supports development and use of shared vocabularies on the Web.

2.2.3 The SPARQL query language

SPARQL (Prud'hommeaux and Seaborne, 2007) is the standard query language for RDF data. A SPARQL query is made up of a set of triple patterns containing a subject, a predicate and an object. Each of the subjects, predicates or objects in a query can be a variable. SPARQL query processor searches for a set of triples that match the triple patterns specified in a query, binding the variables in the query to the corresponding part of each triple. Figure 2.4 shows a SPARQL query that returns all department names and budgets from the table Department.

```
PREFIX vocab: <http://localhostvocab/resource/>
SELECT ?department_name ?budget
WHERE {
   ?department a vocab:department.
   ?department vocab:Department_Department_Name ?department_name.
   ?department vocab:Department_Budget ?budget.
}
```

Figure 2.4: A SPARQL query

SPARQL supports querying semi-structured and ragged data — data in unpredictable and unreliable structure — and querying disparate data sources in a single query. However, it does not support aggregate and group functions. SPARQL is a very young query

language compared to SQL and is still maturing. There are other alternative query languages such as RDF Data Query Language (RDQL) (Seaborne, 2004) and RDF Query Language (RQL) (Karvounarakis et al., 2002).

2.2.4 Examples of Semantic Web projects

I discuss three projects that have made significant contributions to the development of the Semantic Web. However, these projects do not constitute the entire Semantic Web as depicted in Figure 2.2.

FOAF

One of the earliest implementations of Semantic Web application is the Friend of a Friend (FOAF) project (Graves et al., 2007). FOAF creates a web of machine-readable pages that describe people, the links between them and the things that they are interested in. Brickley and Miller (2005) have defined the FOAF vocabulary specification which include the basic classes of entities such as person, organization, group, document and the type of links that exist between these entities. FOAF also takes advantage of Dublin Core (DC) metadata (Powell et al., 2007) for adding semantic annotation to its entities. FOAF continues to solve several problems of identity management on the Web. The University of North Carolina at Chapel Hill has applied FOAF approach to model the structure of its IT department. It is now possible to search staff-related information in seconds (Graves et al., 2007).

DBpedia

The traditional Web is based upon the idea of linking documents through hyperlinks. Semantic Web, on the other hand, is based upon the idea of linking data units. Bizer (2009) shows that links at a lower level of granularity, i.e. data-level, makes it possible to crawl the data space, and provide expressive query capabilities, much like how a database is queried today. Bizer's point is well demonstrated in the DBpedia project (Bizer et al., 2007), a community effort to extract structured information from Wikipedia and make this information accessible in a way users can ask complex questions such as *List all scientists that were born in the 20th century in Canada.* DBpedia knowledge base currently describes more than 2.6 million entities. It has been linked to other data sources on the Web which has made it a central interlinking hub for the emerging Web of data (Bizer et al., 2009).

Kngine

Kngine (ElFadeel and ElFadeel, 2008), known as Web 3.0 search engine, is a semantic search engine designed to understand the meaning of users' queries and return precise results. Depending on the nature of the query, Kngine shows results in visual representation such as graph, comparison table, map, and image. For example, searching for *'3G cellphones'* returns a list of all 3G network compatible mobile phones along with their pictures and relevant details. The results can be filtered by selecting one or more properties of 3G cellphones: year of release, brand name, operating system, camera resolution, and CPU type. Kngine does this by discovering the relationships between the keywords and concepts, and by linking different types of information together.

These examples show how the Semantic Web technologies add new dimensions to the way one can access and use information on the Web. In order to do this at a large scale, the vast majority of the relational data that powers the Web needs to be exposed to Semantic Web structures.

2.3 Building ontologies from relational structures

We have seen in Section 2.1 that one of the major limitations of the relational model is its inability to capture any semantic relationships between data. Section 2.2 shows that Semantic Web knowledge representation techniques can overcome this limitation. Data presented in RDF structures includes the semantic relationships; however it does not capture the domain knowledge that is associated with the data. In order to include the domain knowledge, an evolution from the converted RDF structure into an ontology is required. Relevant research with regards to converting relational to RDF structures is discussed in 2.3.1, and development of full ontologies is discussed in 2.3.2.

2.3.1 Converting relational structure to RDF

There has been a great deal of research on mapping information stored in relational databases to RDF. Sahoo et al. (2009) list two main approaches in mapping relational data to RDF: Extract Transform Load (ETL) mapping and on-demand mapping. ETL process takes relational data as source input and delivers equivalent RDF triples as output. On-demand mapping takes a SPARQL query as input, translates it to an equivalent SQL query, executes the SQL query on relational data, and translates SQL query results to SPARQL query results (in the form of RDF triples) as output. The strengths and

weaknesses of both approaches are summarized in Table 2.2 (Sahoo et al., 2009).

Table 2.2: ETL vs on-demand mapping

	Strengths	**Weaknesses**
ETL mapping	1. Faster query execution 2. Reduced arbitrary performance demand on source RDB	1. Querying large RDF dataset may not be as fast as querying equal amount of RDB data. 2. SPARQL query results may not reflect most recent data. 3. Managing duplicate copies of data in two models.
On-demand mapping	1. Query results are based on most recent data values 2. Data retrieval is based on RDB, and RDB outperforms RDF for analytic queries	1. Arbitrary performance demand on source RDB may affect the performance of legacy information systems

The on-demand mapping is widely preferred method primarily because it allows access to the most up to date information, and it does not burden one with the task of maintaining another version of the same information. An example of on-demand mapping is Virtuoso Universal Server (Erling and Mikhailov, 2007), which converts all primary keys and foreign keys of an RDB into Internationalized Resource Identifiers (IRIs), and assigns a predicate IRI to each column, and *rdf:type* predicate for each row linking it to

an RDF class IRI corresponding to the table. It then takes each column that is neither part of primary or foreign key, and creates a triple consisting of the primary key IRI as subject, the column IRI as predicate, and the column's value as object. This mapping process allows relational data to be rendered as virtual RDF graphs, and accessed using SPARQL queries.

A second example of on-demand mapping is SquirrelRDF (Steer, 2009), a prototype tool that allows SPARQL queries on non-RDF databases such as RDB or Lightweight Directory Access Protocol (LDAP) servers. It automatically generates a mapping file that exposes an RDB schema to an RDF view. Gray et al. (2009) note that the automatically generated RDF views require manual editing for maintaining their referential integrity.

There exist tools that provide both ETL and on-demand mapping services. One of these tools is D2RQ Platform developed by Bizer and Seaborne (2004), which allows one to either convert an entire RDB into a set of RDF triples, or access an RDB as virtual and read-only RDF triples. D2RQ Platform consists of two main components: *D2RQ Engine* and *D2R Server*. D2RQ Engine relies on Jena (McBride et al., 2010) and Sesame (Broekstra et al., 2002), which are frameworks for storing and querying RDF data. D2RQ Engine works as a plug-in for Jena or Sesame Semantic Web toolkits by rewriting Jena or Sesame API calls and SPARQL queries to SQL queries using its D2RQ mapping language. The results of these SQL queries are then transformed into RDF triples and passed onto the D2R Server for publishing on the Semantic Web.

D2RQ performance was compared to the performance of Jena2 database back end using a dataset of 200,000 paper descriptions from the DBLP Computer Science Bibliography. Jena2, which is a subsystem of Jena, stores RDF triples using a relational database. Query execution time was measured in milliseconds. The **find(s p o)** query was run on both platforms. This query is a minimal SPARQL query used for experimental purpose. The parameter '**s**' represents **the subject**, '**p**' represents **the predicate** and '**o**' represents **the object**. The '**?**' parameter implies **any** for matching that slot. For instance, if '**s**' denotes 'books', '**p**' denotes 'has authors' and '**o**' denotes 'authors names', **find(? ? Tanenbaum)** will return all books authored by Tanenbaum. The results from the performance comparison test by Bizer and Seaborne (2004) are shown in Table 2.3.

Table 2.3: Performance comparison between Jena2 and D2RQ Platform

find(s p o) query	Jena2	D2RQ
1. find (s ? ?)	1.83 ms	0.01 ms
2. find (? p o)	1.94 ms	0.97 ms
3. find (? p ?)	42431 ms	72 ms
4. find (? ? o)	1.72 ms	3.23 ms

As seen in the performance results, D2RQ executes queries much faster than Jena2 implementation. Sequeda et al. (2008), however, notes that in D2RQ approach, mapping between a relational schema and existing ontology requires one to manually specify the classes and the hierarchies between classes using an ontology editor. In addition,

Bizer and Cyganiak (2007) listed a number of limitations of D2RQ platform. It does not support integration of multiple RDBs or other data source; it does not allow data manipulation; and it does not provide any inference capability.

SquirrelRDF and D2RQ Platform still remain prototype tools for RDB to RDF mapping. Gray et al. (2009) prove that these prototypes fail to expose large science archives stored in relational format. The authors have tested several RDB to RDF mapping tools including D2RQ and SquirrelRDF for executing queries over a sample of large astronomical data set and have come to the conclusion that more research and improvements are required for SPARQL and RDB to RDF mapping tools for exposing science archive data. They have tested with 18 standard scientific SQL queries and only 9 of them can be expressed in SPARQL queries. SPARQL also does not support mathematical functions such as aggregate and trigonometric functions.

The mapping tools and techniques that have been discussed so far are intended for converting data from relational structure to RDF structure. Semantic Web researchers have also attempted to create tools for converting an RDB schema to an ontology. For instance, Sequeda et al. (2009) created Ultrawrap, an automatic wrapping system that generates an OWL ontology from RDB schema. The authors, however, doubt whether the results of a purely syntax driven translation of an RDB schema to OWL can qualify for a comprehensive ontology. Therefore, they use the term *putative ontology* to describe the resulting OWL ontology. The putative ontology works as a basis for the user to write SPARQL queries. Ultrawrap then natively translates SPARQL queries to SQL queries and uses an SQL optimizer to execute the SQL queries on the RDB system. It does not

provide any reasoning capabilities.

Mapping a relational database to an ontology is a challenging task. Cullot et al. (2007) have developed a prototype tool called DB2OWL to create an ontology from a relational database. Its mapping process classifies RDB tables into three different cases to determine which ontology structures are to be created from which database components. Tables in RDB are mapped to OWL classes and sub-classes. RDB table cases determine whether a table is mapped to an OWL class or sub-class. Then RDB columns are mapped to OWL properties. Primary key and foreign key relationships are mapped to object properties in order to preserve their referential integrity. During the mapping process, a mapping file is generated and used to translate ontological queries into SQL queries and retrieve corresponding instances.

The previous examples mostly show general mapping of relational to RDF structure regardless of the domain of the data. There has also been research in building tools for mapping domain-specific data. For example, Byrne (2008) shows a general mechanism for converting cultural heritage data from relational databases to RDF triples. The author created a triplestore – specialized database for the storage and retrieval of RDF triples – named *Tether* from the Royal Commission on the Ancient and Historical Monuments of Scotland (RCAHMS) database of around 250,000 historical sites with 1.5 million archives and bibliographic materials.

2.3.2 Upgrading converted structures to full ontologies

The previous section discussed techniques for converting relational to RDF structures from which a list of terms and their properties can be extracted. However, the converted structures do not capture the inherent knowledge in the data that often resides in a data dictionary or in the mind of the DBA. In order to capture the domain knowledge, the converted structures need to be upgraded into full ontologies. Noy and McGuinness (2001) state that there is no prescribed way or methodology for developing ontologies; the best solution always depends on the application in mind and the extensions that follow. The authors recommend the following seven guiding steps in developing an ontology:

Step 1: Determine the domain and scope of the ontology

The development process of an ontology starts with the definition of its domain and scope. In this step, the developer can ask some basic questions such as: (i) What is the domain that the ontology will cover? (ii) For what purpose the ontology is going to be used? (iii) For what types of questions the information in the ontology should provide answers to? (iv) who will use and maintain the ontology? The answers to these questions help determine the domain and limit the scope of the ontology.

Step 2: Consider reusing existing ontologies

The developer should check what others have done and whether it is possible to refine and extend existing ontologies instead of creating from the scratch. Reusing ontologies is important because creating an ontology for each application defeats the purpose of sharing knowledge. There are ontologies that are publicly available on the Web such as Ontolingua Server (2008) and DAML Ontology Library (2004),

which can be imported into an ontology development environment.

Step 3: Enumerate important terms in the ontology

Create a list of all terms that one would like to make statements about. The terms can be formulated by asking some basic questions such as: (i) What are the terms that would one like to talk about? (ii) What properties do these terms have? (iii) what would one like to say about these terms? It is important to create a comprehensive list of all terms.

Step 4: Define classes and the class hierarchy

There are three approaches in defining a set of classes: a *top-down* development process starts with the definition of the most general concepts in the domain followed by subsequent specialization of the general concepts; a *bottom-up* development process starts with the definition of the most specific concepts followed by subsequent grouping of these concepts into more general concepts; and a *combination* development process is a blend of top-down and bottom-up processes which starts with the definition of more notable concepts and then proceeds with appropriate generalization and specialization of the remaining concepts.

Whichever approach is followed, one should start by defining classes. From the terms listed in Step 3, ones that describe concepts having independent existence are defined as classes in the ontology. The classes are then organized into a hierarchical structure. The class hierarchy represents an 'is-a' relation: a class A is a subclass of B if every instance of A is also an instance of B.

Step 5: Define the properties of the classes – slots

The properties of a class describe the internal structure of the class. Some of the terms formulated in Step 3 are defined as classes in Step 4; most of the remaining terms are then defined as properties of those classes. For a given property, one must determine which class it describes. Thus properties become slots attached to their respective classes. All subclasses inherit the slots of its parent class. For example, if a slot called *Last name* is added to the class *Person*, and *Student* is a subclass of *Person*, then *Student* will inherit the slot *Last name*.

Step 6: Define the facets of the slots

A slot can have different facets describing the value types, allowable values, cardinality, etc. Value types describe what types of values can fill in the slot. Common value types are string, number, and boolean. Slot cardinality defines what is the minimum and maximum number of values a slot can have.

Step 7: Create instances

In the final step, individual instances of each class are created. This is done by first, choosing a class; second, creating an individual instance of that class; and third, filling in the slot values.

The ontology development process should not stop with completing the final step. Rather it should be an iterative one in which a basic ontology is first created, and then revised and refined to fill in the missing pieces.

There are a variety of languages for developing ontologies. RDF Schema (RDFS), which is an extension of the RDF language, introduces basic ontological primitives such as class, subclass, domain, range, etc that are used to define concepts and their relationships in an ontology. The W3C has adopted OWL (McGuinness and Harmelen, 2004), which uses RDF and XML syntax and provides Description Logic (DL) based reasoning support, as the standard ontology language for the Semantic Web. McGuinness and Harmelen provide a comprehensive list of OWL primitives. The main primitives are summarized in Table 2.4.

There exist a number tools for developing and managing ontologies. Protégé (Horridge et al., 2009) is an open source ontology editor which allows one to create and export ontologies into various formats including RDFS and OWL. Fensel et al. (2001) extended RDFS to Ontology Inference Layer or Ontology Interchange Language (OIL), an ontology infrastructure which includes the definition of a formal semantics based on DL, an ontology editor, and inference engines for reasoning capabilities.

Protégé and OIL are widely used tools for authoring ontologies from scratch or modifying existing ontologies; however, they do not provide any integrative function to work with an RDB to RDF mapping tool. TopBraid Composer (TBC) (TopQuadrant Inc., 2007) provides integrative function for connecting to an RDB through D2RQ Platform. TBC is an application development framework which provides a comprehensive set of tools covering the life cycle of semantic application development. TBC's integrative feature allows one to import an RDB structure into TBC as a base ontology and modify it towards building a more comprehensive ontology. RDB table names are represented

Table 2.4: The main primitives of OWL

owl:class: A class is a group of entities that share some common characteristics. Classes can be organized in a hierarchical order ranging from general to specialized classes. In class hierarchy, a general class is known as parent class, and a special class is called subclass.
rdfs:subClassOf: A specialized class can be defined as a subclass of its parent class. For example, the class *Student* can be stated as a subclass of *Person*. This subclass definition allows a reasoner to deduce that if an entity is a Student then it is also a Person.
rdfs:property: A Property asserts general facts about the members of classes and specific facts about individuals. There are two types of properties: *datatype property* and *object property*. Datatype properties are relations between instances of classes and RDF literals and XML Schema datatypes, whereas Object properties are relations between instances of two classes.
rdfs:subPropertyOf: Property hierarchies can be created by stating that a property is a subproperty of another property.
rdfs:domain: A domain of a property limits the individuals to which the property can be applied. If a property relates an individual to another individual, and the property has a class as one of its domains, then the individual must belong to the class.
rdfs:range: The range of a property limits the individuals that the property may have as its value. If a property relates an individual to another individual, and the property has a class as its range, then the other individual must belong to the range class.
rdf:ID: The instances of classes are declared using this primitive.
owl:equivalentClass: Two classes may be stated as equivalent classes if they have the same instances. Equality can be used to create synonymous classes.
owl:equivalentProperty: Two properties may be stated as equivalent properties. Equality may be used to create synonymous properties.
owl:sameAs: Two instances may be stated to be the same. An instance can be identified by a number of different names using this primitive.

as classes and column names are represented as datatype properties. Primary key and foreign key relationships are represented as object properties. One can extend existing classes with superclasses and subclasses and specify the properties that the subclasses

inherit from the superclasses. Ontologies created using TBC also can be exported to RDFS and OWL format.

Ontology language use certain types of logic to support reasoning. A reasoning engine can infer logical consequences from a set of asserted facts and the inference results vary depending on which of the two assumptions are in place. The Closed World Assumption (CWA) states that "databases (and people) assume that the information provided is complete, so that ground atomic sentences are not asserted to be true are assumed to be false" (Russell and Norvig, 2003). For example, assuming that a student database contains information about all students, if the name 'Adam' is not found in the database, a reasoning engine will conclude that 'Adam' is not a student. The CWA is often complemented by the Unique Name Assumption (UNA) which assumes that names in a knowledge-base are unique and refer to distinct instances. The Open World Assumption (OWA), on the other hand, assumes that the descriptions of resources are not confined to a single knowledge-base or scope (Smith et al., 2004). In other words, from the absence of a statement alone a reasoner cannot conclude that the statement is false.

OWL is designed with the purpose of defining Web ontologies. The Web is an open and dynamic environment in which information continues to evolve, and at any point in time one cannot assume its completeness. Therefore, the OWA is more appropriate while reasoning with information presented on the Web. Ricca et al. (2009) argue that OWL's OWA is unsuited for modeling enterprise ontologies because they evolve from relational databases where both CWA and UNA are mandatory. In addition, the presence of naming conventions in most enterprises can guarantee uniqueness of names which makes the

UNA relevant. The authors have proposed an ontology representation language called OntoDLP which extends Answer Set Programming (ASP) with the main features such as classes, inheritance, relations and axioms that are relevant to ontologies. ASP is a kind of logic programming with negation as failure that works by translating the logic program into ground form and then searching for answer sets (Russell and Norvig, 2003). OntoDLP is used by OntoDLV, a system that facilitates specification and reasoning of enterprise ontologies.

The mapping tools and techniques that I have presented in subsection 2.3.1, can fully automate the process of converting relational to RDF structure. Beyond this point, there is little or no automated assistance for upgrading the converted RDF structures to full ontologies. The available tools for upgrading RDF structures to ontologies require human expertise with considerable understanding of the ontology development process.

2.4 Multiagent systems

Wooldridge (2009) defines agent as "a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its delegated objectives". Although an agent system may operate alone in an environment and when necessary interact with its users, in most cases they consist of multiple agents. These multiagent systems can model complex software systems in which individual agents interact and collaborate to achieve a common goal or compete to serve their self interests (Bellifemine et al., 2007). Wooldridge and Jennings (1995) suggest three capabilities of an intelligent agent:

1. *Reactivity:* Intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives.
2. *Proactiveness:* Intelligent agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives.
3. *Social Ability:* Intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

In order to achieve these capabilities, agents are modeled with the mental abilities such as beliefs, desires, and intentions. This is due to the fact that humans use these concepts as an abstraction mechanism for understanding the properties of a complex system. Developing machines with such mental qualities was first proposed by Mccarthy (1979). Shoham (1993) then articulated the idea of programming software systems in terms of mental states.

The remainder of this section is organized as follows: agent-oriented software engineering is discussed in Section 2.4.1; how the Semantic Web relates to agents is presented in Section 2.4.2; and issues related to agent-human interactions are discussed in Section 2.4.3.

2.4.1 Agent-oriented software engineering

Agent-oriented approach in Software Engineering is a new software paradigm that models a complex system as a collection of autonomous, proactive, and social agents. Shoham (1993) first introduced the concept of Agent Oriented Programming (AOP) as a new software development paradigm which can be viewed as a specialization of Object Oriented

Programming (OOP). Though both paradigms may appear similar from the theoretical perspective, they have visible differences. Wooldridge (2009) lists three distinctions between AOP and OOP. First, agents exhibit autonomous behaviors while objects depend on external invocation. Agents enjoy the freedom to make their own decision whether or not to perform an action. Second, agents are reactive, proactive, and social, whereas the object-oriented model has nothing to do with these behaviors. Third, in a multiagent system, each agent is considered to have its own thread of control whereas in a standard object-oriented model the system has a single thread of control.

Jennings and Wooldridge (2000) argue why agent-oriented techniques are well-suited to developing complex software systems. The authors compared agent-oriented techniques with the object-oriented approach. In the object-oriented approach, an object perform its actions only when it is instructed by an external invocation. This approach may work for smaller application in cooperating and well-controlled environments; however, it is not suited to complex or competitive environments because it gives the control to execute an action to the client requesting that action and not the action executor. Thus objects are obedient to one another. Agent-oriented approaches allow the action executor – the agent – to decide whether or not to perform an action because it is more intimate with the details of the actions to be performed, therefore, it may know a good reason for executing or refusing to perform an action. The Object-oriented approach also fails to provide an adequate set of mechanisms for modeling a complex system that comprises of inter-related subsystems. Agent-oriented techniques provide problem solving abstraction for modeling the dependencies and interactions that exist in such complex systems.

There are a number of platforms available for developing multiagent systems such as Jason (Bordini et al., 2007), Jadex (Pokahr et al., 2005), and JADE (Bellifemine et al., 2007). Jason, which is a Java based platform, uses AgentSpeak agent-oriented programming language to program the behavior of individual agents. Jadex is a popular open source platform for programming intelligent software agents using XML and Java. JADE is an agent-oriented middleware that provides domain-independent infrastructure for developing multiagent systems. *Telecom Italia* distributes Jade as open source software under the terms of the LGPL (Lesser General Public License) Version 2.

2.4.2 Agents and Semantic Web

Berners-Lee et al. (2001) envision a Semantic Web agent that communicates with other agents to set up a doctor's appointment.

> At the doctor's office, Lucy instructed her Semantic Web agent through her handheld Web browser. The agent promptly retrieved information about Mom's *prescribed treatment* from the doctor's agent, looked up several lists of *providers*, and checked for the ones *in-plan* for Mom's insurance within a *20-mile radius* of her *home* and with a *rating of excellent* or *very good* on trusted rating services. It then began trying to find a match between available *appointment times* (supplied by the agents of individual providers through their Web sites) and Pete's and Lucy's busy schedules. (The emphasized keywords indicate terms whose semantics, or meaning, were defined for the agent through the Semantic Web.)

As information on the Semantic Web is presented with semantic annotations and in the form on ontologies, agents are able to understand the meaning of this information and take appropriate actions based on perceived meaning. Thus, the entire Web becomes part of agents' environment in which agents can proactively search for necessary information to serve its intended goals. However, realizing Berners-Lee's grandiose vision of Semantic Web agents has been highly challenging due to the dynamic and heterogeneous nature of the Semantic Web (Tamma and Payne, 2008). The authors have formulated some challenges faced by Semantic Web agents. These challenges can be summarized as follows: discovering resources; determining ontology identity; ontology reconciliation; dynamic evolution of agent ontologies; describing dialogues and protocols using ontologies; and representing and reasoning with uncertain information.

The W3C in collaboration with Semantic Web researchers continues to develop tools and standards that may overcome some of the challenges identified by Tamma and Payne.

2.4.3 Human-agent interactions

Human-agent interactions can be regarded as a specialization of human-computer interactions. So far, there is no standard language or communication mechanism for humans to interact with agents. This choice is left with the designer to decide how agents are instructed and in which format agents report back to the users.

Of particular interest is the issue that when it comes to communicating with others, humans regard certain values such as speaking manner very highly. Agents, when interacting with humans should also follow a certain code of communication ethics. Brad-

shaw et al. (2011) outline a number of characteristics of a good agent with regard to joint activity in the following maxims:

- A good agent is observable. It makes its pertinent state and intentions obvious.
- A good agent is attuned to the requirement of progress appraisal. It enables others to stay informed about the status of its tasks and identifies any potential trouble spots ahead.
- A good agent is informative and polite. It knows enough about others and their situations so that it can tailor its messages to be helpful, opportune, and appropriately presented.
- A good agent knows its limits. It knows when to take the initiative on its own, and when it needs to wait for outside direction. It respects policy-based constraints on its behavior, but will consider exceptions and workarounds when appropriate.
- A good agent is predictable and dependable. It can be counted on to do its part.
- A good agent is directable at all levels of the sense-plan-act cycle. It can be retasked in a timely way by a recognized authority whenever circumstances require.
- A good agent is selective. It helps others focus attention on what is most important in the current context.

- A good agent is coordinated. It helps communicate, manage, and deconflict dependencies among activities, knowledge, and resources that are prerequisites to effective task performance and the maintenance of common ground.

The set of characteristics that makes an agent good or bad is a subjective choice. What is considered good manners in one culture may not be the case in another culture. Therefore, a designer should take into account the cultural sensitivity that agents may experience while engaging with other agents and human users.

Chapter 3

Semantic Query Access to Legacy Relational Databases using Intelligent Middleware

In this thesis, I propose and investigate a novel approach to accessing information stored in legacy relational database (RDB) systems. This approach is motivated by the rising demand for flexible access to information through user-friendly interfaces for human users as well as standardized software interfaces for intelligent agents that act on their behalf. The Semantic Web project envisions a world-wide infrastructure providing such services through the adoption of associated standards, and the use of tools and ontologies that are being developed towards realizing the objective of the Semantic Web. In that context, I explore how an institutional decision support system, based on legacy RDB systems, could employ a combination of Semantic Web and multiagent systems (MAS) technologies to evolve towards flexible semantic access to information within its own operational

scope.

Legacy RDB systems are typically used for decision support purposes by a limited number of users with considerable knowledge about the domain of the information stored in relational databases. The users explain their information requirements to report writers through natural language communication, which refers to domain knowledge that is not explicitly captured within the database itself. The report writers use their acquired domain knowledge and technical expertise to translate the users' requests into formal SQL queries for extracting the necessary information. Thus, the users of legacy RDB systems always depend on report writers for bridging the gap between domain-level discourse and RDB query.

Software agents can assist human users in a decision support environment by reasoning with information underlying the decision process; however, this requires the information to be structured using ontologies that formally capture the domain knowledge associated with that information. By developing the necessary ontologies within the system itself, and by introducing a simplified natural language interface, the proposed approach enables the user to directly access information without depending on the assistance of human intermediaries such as report writers. This implies that the structures of the information stored in RDB systems need to be represented in the form of ontologies. As such, I explore the possibility of converting an RDB schema to ontologies while keeping the RDB system's original design and functionalities intact. An RDB schema can be converted to ontologies in two steps: first, transforming the RDB schema to an RDF structure; and second, upgrading the RDF structure to an ontology. The first step has been fully

automated by several RDB-to-RDF conversion tools, namely Virtuoso Universal Server, SquirrelRDF, and the D2RQ Platform. The second step requires human involvement for manually upgrading the converted structures to ontologies using ontology editors such as Protégé, TopBraid Composer, and OntoDLV.

To overcome some of the issues raised above I propose a system architecture, called the Semantic Report Generation System (SRGS). In defining the architecture of SRGS, I focus on the following tasks:

1. Develop a definition of a distributed system architecture that combines Semantic Web and MAS technologies in an intelligent middleware layer, which supports the building of ontologies from RDB structures, and provides the user with effective semantic query access to information in RDB systems.

2. Examine to what extent one can realize such a system architecture using the existing software systems that have already been developed or envisioned in the context of the Semantic Web and MAS research.

Tasks one and two are carried out in an iterative manner in which I identify existing software systems that can be integrated as components in the new system architecture and then refine the architectural definition so that it can rely on the identified components.

The proposed system will allow the Database Administrator (DBA) to incrementally build ontologies from an RDB schema with the assistance of a software agent. The architecture takes advantage of the existing automated process for converting RDB schema

to RDF structure, and introduces a software agent to assist the DBA in developing a *reference ontology* from the RDF structure, human domain knowledge, and knowledge resources on the Semantic Web. Another software agent assists the user to develop a *custom ontology*, which defines user-specific concepts using entries from the reference ontology. In this approach, the agents interact with human actors throughout the ontology development process. The agents perform some of the technical tasks while the human actors make decisions. This role in ontology development adds a new dimension to the traditional user and DBA profiles. However, this does not require them to become technical experts fully specialized in the ontology development process because the agents are responsible for executing some of the technical tasks.

The ontologies create a layer of semantic information structures on top of the existing legacy RDB system that enables semantic queries and allows agents to reason about the information stored in the RDB. In addition, the system includes a simplified natural language interface which enables the users to directly communicate their requests for information stored in the underlying RDB systems without depending on any human intermediaries. In this process, a software agent assists the users to formulate requests for information using the simplified natural language.

Chapter 4

The Architectural Model

In defining the architecture of Semantic Report Generation System (SRGS), I begin by abstracting the aspects of the system that are relevant to my study topic. I am primarily interested in the viewpoint of the user who accesses information in an existing relational database (RDB) system. The structure of the database and its contents are not controlled by the user. The user is aware that the structure and contents of the database may evolve over time. However, the requirements that cause such changes to be made are beyond the scope of my current interest. If the present user can influence such requirements, those influences occur outside of my model. I assume that the user has some knowledge about the domain as well as the source of information.

The system requirements are described in Section 4.1; the global architecture of the system is presented in Section 4.2; agent roles are discussed in Section 4.3; and incorporation of existing Semantic Web and MAS components is discussed in Section 4.4.

4.1 The system requirements

The system requirements are developed in three steps. Subsection 4.1.1 describes the requirements for a generic system that represents the basic functionality of user access to information in an RDB system in a way that is common to its many possible implementations. The focus of 4.1.2 is on user-system interaction in legacy RDB system. Subsection 4.1.3 describes the requirements for user-system interaction for SRGS.

The system requirements are represented in the form of use cases and actors. Rumbaugh et al. (2004) define use case as a coherent unit of functionality expressed as a transaction among actors and the system. An actor may be a person, organization or other external entity that interacts with the system.

4.1.1 The generic system

The actors and high-level use cases of the generic system are shown in Figure 4.1.

The actor of primary interest is the User. The other two actors, the Database Administrator (DBA) and the Data Entry Operator (DEO), maintain the RDB structure and content respectively.

The top four use cases of Figure 4.1 capture the generic system functions performed on behalf of the user, regardless of how these functions are executed. For example, in a legacy system the user typically performs these functions through a human intermediary who in turn accesses the computer system. In SRGS, the user performs the same functions through direct interaction with the computer system.

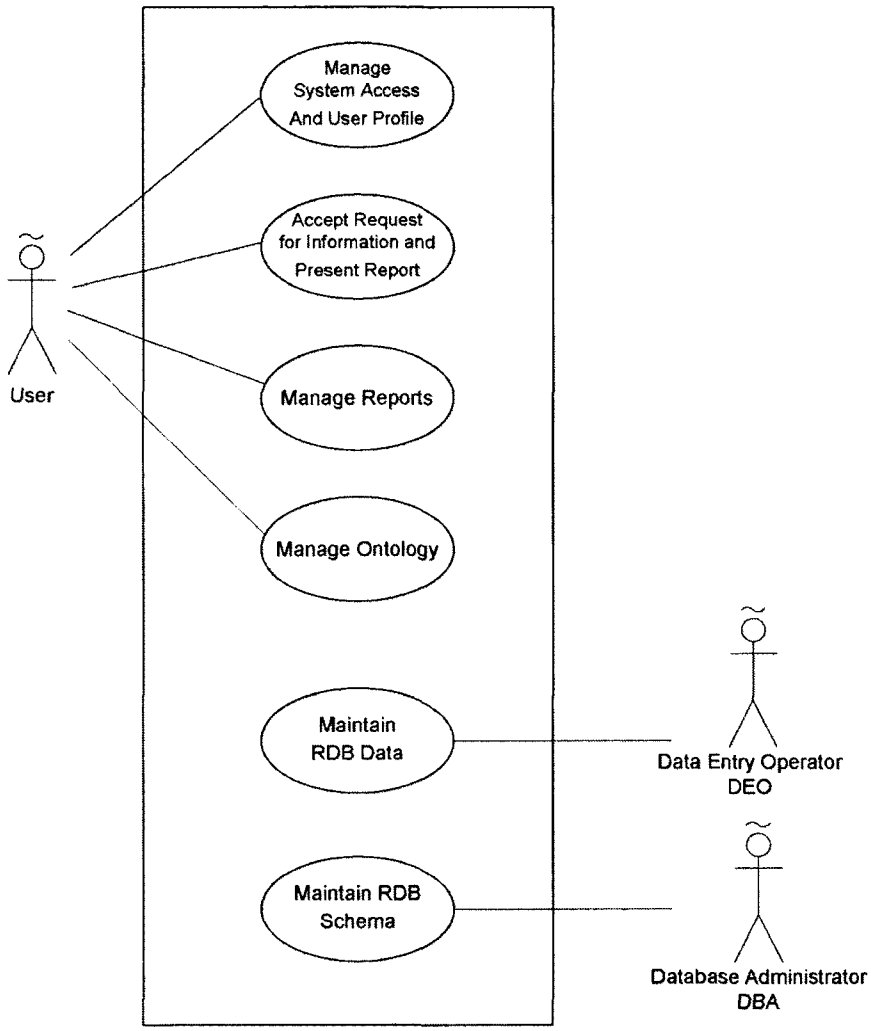

Figure 4.1: The actors and high-level use cases of the generic system

Manage System Access and User Profile

System access is based on user authentication, which verifies the user's identity and specific access rights. The user can also specify a set of preferences contained in the user profile with regards to the various options available in the user interface. This general use case includes system help and tutorial assistance.

Accept Request for Information and Present Report

In this use case, the system accepts request in which the user specifies what information should be retrieved and how it should be presented; the system then retrieves the information and presents it in the requested format.

Manage Reports

This use case allows the user to save, delete, reformat, and retrieve reports.

Manage Ontology

As the usage of data and the environment evolve there is a need to introduce new terms and modify some definitions of terms in the ontology. The ontology itself consists of two components. The first one, which I call the *reference ontology*, is incrementally developed from the structure of the RDB. It describes concepts and the semantic relationships between the concepts in an application domain. The second component, called the *custom ontology* describes the conceptual framework specific to a particular user, that can be directly translated to the reference ontology. Modifications of the reference ontology occur when the DBA changes the structure of the database, or when the represented knowledge is updated due to external factors, such as changes in the organizational policies. Modifications of the custom ontology mainly occur when the user introduces new concepts and their relationships that are defined using constructs in the reference ontology.

Maintain RDB data

This use case allows the DEO to insert, delete, and modify data in the RDB system.

Maintain RDB schema

This use case enables the DBA to modify the structure of the RDB system. When necessary, the DBA may add, remove, or change RDB tables and the columns withing the tables.

4.1.2 Legacy RDB system

In a legacy RDB system, some of the functionalities of the generic system shown in Figure 4.1 are performed by the human intermediary report writer on behalf of the user while the other functionalities are performed by the system. The DBA and the DEO actors play the same roles as in a generic system, and the use cases are executed in similar ways by the system. The actors and the high-level use cases of legacy RDB system are shown in Figure 4.2.

Below I briefly describe the four use cases from the perspective of a legacy RDB system.

Manage System Access and User Profile

The user delegates access right to the report writer who accesses the system. System access is based on user authentication, which verifies the report writer's identity, and authorization. This general use case may include system help and tutorial assistance.

Accept Request for Information and Present Report

In this use case, the report writer accepts a request, in which the user specifies what information should be retrieved and how it should be presented; the report

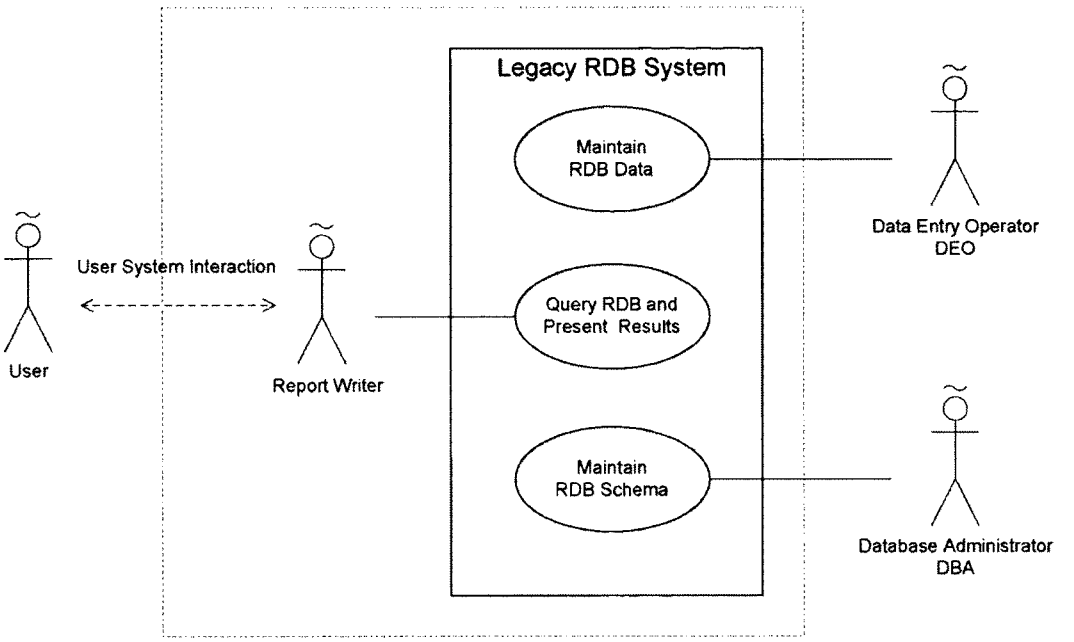

Figure 4.2: Legacy RDB system

writer then queries the RDB to retrieve the information and presents it to the user in the requested format. If the request is not clear, the report writer engages with the user to further clarify the request through natural language communication.

Manage Reports

This use case allows the report writer to save, delete, reformat and retrieve reports on behalf of the user. The user typically receives printed copies of the report.

Manage Ontology

The management of the reference ontology occurs between the report writer and the DBA; the management of the custom ontology occurs between the user and the report writer. The ontologies represent knowledge and personal experience of

the actors informally recorded in electronic and paper-based documents or simply remembered by the actors. Managing both ontologies involve natural language communications between the actors. For instance, the report writer informs the user about modifications of concepts in the reference ontology; the user informs the report writer about a new concept the user wants to introduce to the custom ontology.

Maintain RDB data

This use case provides the same services as specified in the generic system.

Maintain RDB schema

This use case provides the same services as specified in the generic system.

This process of negotiation and delegation between the user and the report writer is often time consuming, resulting in delays and costs.

4.1.3 The Semantic Report Generation System (SRGS)

In SRGS, the user directly interacts with the system that performs the functionalities in the top four use cases shown in Figure 4.1. The short descriptions of the use cases are as follows.

Manage System Access and User Profile

System access is based on user authentication, which verifies the user's identity, and authorization. The user-system communication occurs in an interactive manner. The user can specify personal preferences for communicating with the system. The

system helps the user to specify personal preferences and manages them. This general use case includes system help and tutorial assistance.

Accept Request for Information and Present Report

This use case allows the user to directly communicate report requests to the system. In the request, the user specifies what information should be retrieved and how it should be presented. If the user's request is not clear, the system asks the user to further clarify the request. This clarification process is an interactive one in which the system ensures that it understands the user's request. The system tries to mimic the role of the report writer in legacy RDB system. It then retrieves the information and presents in the requested format.

Manage Reports

This use case allows the user to save, delete, reformat and retrieve reports.

Manage Ontology

In SRGS, the reference ontology formally represents knowledge originating from the underlying relational database, human actors in the system, and ontological knowledge available on the Semantic Web. The DBA interacts with the system in building and maintaining the reference ontology. Thus, the DBA's actor profile now includes the new role of managing the reference ontology in addition to the traditional role of managing RDB systems. The custom ontology allows the user to introduce user-specific concepts and their relationships on top of the reference ontology. The user now has the additional role of managing the custom ontology.

Maintain RDB data

This use case provides the same services as specified in the generic system.

Maintain RDB schema

This use case provides the same services as specified in the generic system.

The functions of each high level use case can be further specified by decomposing into simpler use cases. I discuss decomposition of the following two high-level use cases.

1. Accept Request for Information and Present Report

The decomposition of the *Accept Request for Information and Present Report* use case is shown in Figure 4.3. The decomposed uses cases are grouped by their functionalities into the use cases that directly communicate with the user: the *Front End*, and the use cases that communicate with the legacy RDB system: the *Back End*. The Front End and the Back End can reside on different machines and communicate through a network. In general, a Back End can support multiple Front Ends, and a Front End can interact with multiple Back Ends.

In the Front End, the *Accept Request for Information (SNL) and Present Results* use case allows the user to formulate request for information in a Simplified Natural Language (SNL). The request contains domain-specific terms that specify the information to be retrieved, and keywords that describe the format for presenting the retrieved information. Once the request is accepted, the *Parse and Interpret SNL Request* use case produces an intermediate representation of the user request, and the *Verify Request Ontology* use case ensures that each statement as a whole in the request is semantically correct. If the

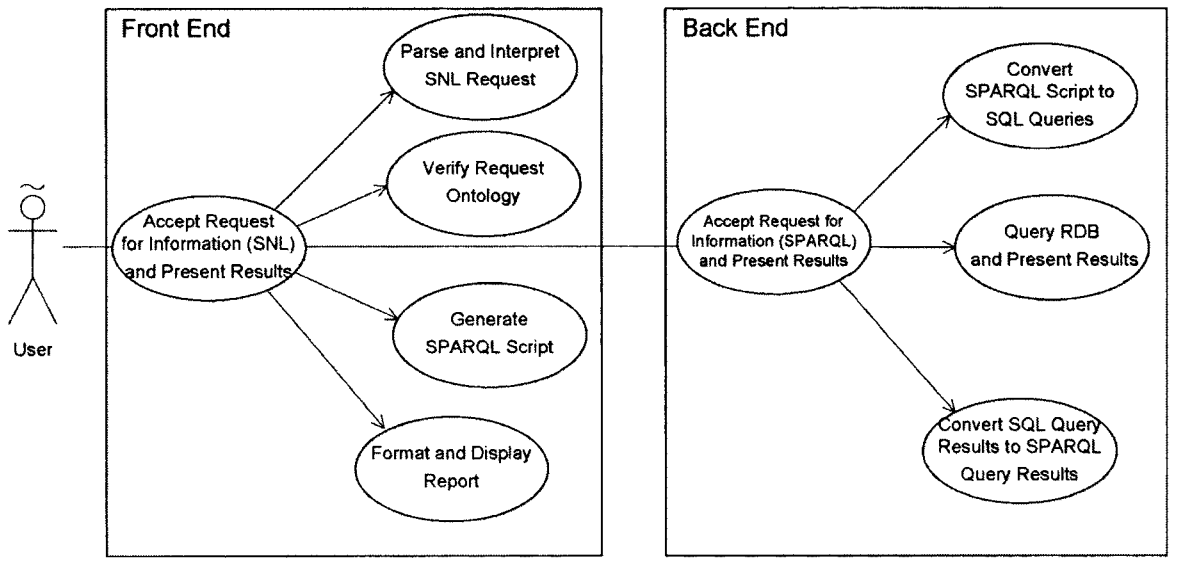

Figure 4.3: Accept request for information and present report

SNL request is valid, the *Generate SPARQL Script* use case creates a SPARQL script from the intermediate representation of the SNL request. The Front End then sends the SPARQL script to the Back End for further processing. The Front End receives the SPARQL results sent from the Back End. The SPARQL results are then formatted as report and presented by the *Format and Display Report* use case.

In the Back End, the *Accept Request for Information (SPARQL) and Present Results* use case receives the SPARQL script and translates it to equivalent SQL queries by the functions in the *Convert SPARQL Script to SQL Queries* use case. The *Query RDB and Present Results* use case then executes the SQL queries on the RDB system and sends the SQL results to the *Convert SQL Query Results to SPARQL Query Results* use case, which translates the SQL query results to SPARQL results. The *Accept Request for Information (SPARQL) and Present Results* use case sends the SPARQL results to the

Front End.

2. Manage Ontology

The decomposition of the *Manage Ontology* use case is shown in Figure 4.4. The use cases are grouped by their functionalities into use cases that directly communicate with the user, the *Front End*; and use cases that communicate with the DBA, the *Back End.*

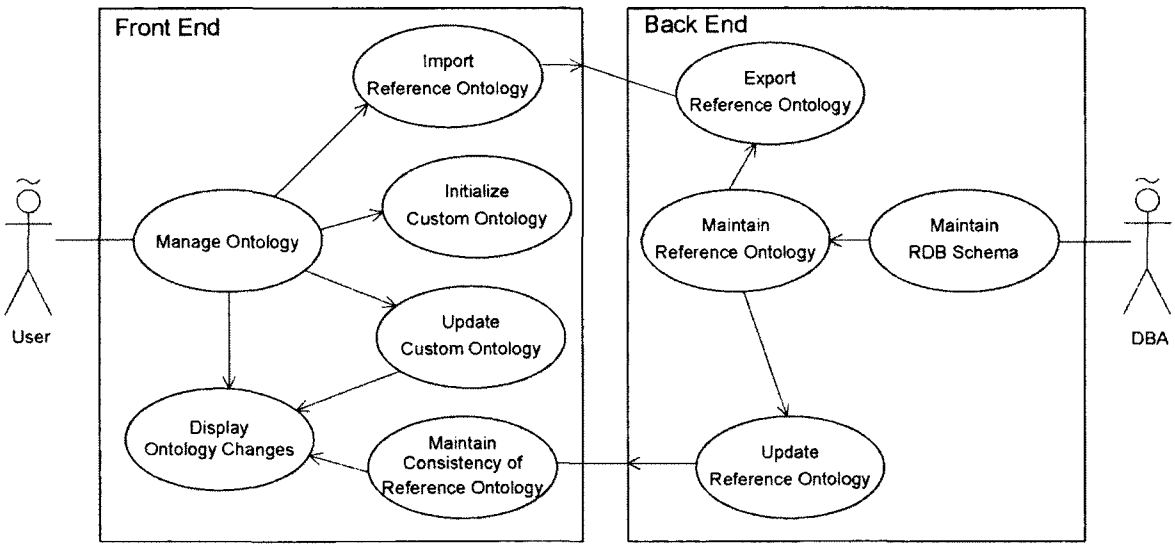

Figure 4.4: Manage ontology

The Front End initially imports the reference ontology when it connects to the Back End. While importing the reference ontology, the functions of the *Import Reference Ontology* use case in the Front End rely on the functions of the *Export Reference Ontology* use case in the Back End. The *Initialize Custom Ontology* use case allows the user to create a conceptual framework specific to the user. Through the functions in the *Update Custom Ontology* use case, the user can modify the definition of user-specific concepts

in the custom ontology. When the reference ontology is updated in the Back End, the *Maintain Consistency of Reference Ontology* use case ensures that the updates are also applied to the reference ontology in the Front End. The reference ontology updates are displayed to the user by the *Display Ontology Changes* use case.

In the Back End, the *Export Reference Ontology* use case sends a copy of the reference ontology to the attached Front End. The *Maintain RDB Schema* use case allows the DBA to modify the structure of the RDB by changing the RDB schema. When the DBA changes the RDB schema, the *Maintain Reference Ontology* use case incorporates the schema changes into the reference ontology with the help of the *Update Reference Ontology* use case.

4.2 The multiagent architecture of SRGS

At the high level, SRGS consists of two subsystems: *User Subsystem (US)* and *Database Subsystem (DBS)*. The primary functions of the US include accepting the user's request for information, presenting reports, and developing the custom ontology. The DBS is responsible for retrieving information from the legacy RDB system and developing the reference ontology. In the basic architecture, each subsystem is comprised of an agent and an environment which contains several software components. The agent perceives the behaviors of the components in the environment and influences their future actions. Subsection 4.2.1 describes the basic architecture of the system in which a single user is connected to the US which is attached to one DBS. Subsection 4.2.2 through 4.2.5 present three different versions of the system architecture with regards to the multiplicity of users and the subsystems.

4.2.1 The basic architecture

The basic architecture is described using a very simple configuration of SRGS. It consists of a single US and a single DBS, connected through a wide area network, with a single user accessing the system. This configuration is depicted in Figure 4.5.

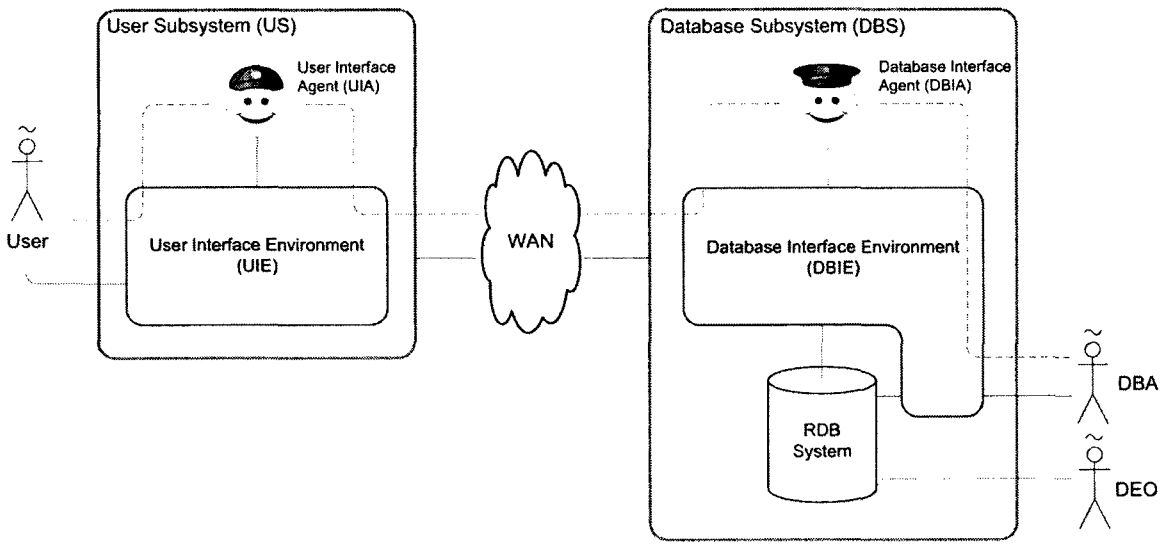

Figure 4.5: Single User Subsystem to single Database Subsystem

The User Subsystem

The architectural structure of the US is shown in Figure 4.6. The User Interface Environment (UIE) comprises the components that provide the main subsystem functions. The primary purpose of the UIE is to execute the routine user requests efficiently, without the need to engage in reasoning in the sense of artificial intelligence techniques. The UIE components can be designed and implemented using the conventional Object-Oriented Software Engineering (OOSE) methodology, with an emphasis on efficient performance.

The User Interface Agent (UIA) can observe the events in the environment, including the behavior of individual components, and act on the environment to influence the behavior of its components. The agent provides the practical reasoning (i. e., deliberation and planning) capabilities to the subsystem, enabling it to autonomously resolve arising problems without intervention of human experts. Its presence introduces the qualities of flexibility, adaptability, tolerance to variations in user preferences and practices, and evolution of the subsystem behavior according to changing user requirements. Those qualities are necessary in order for the system to meet the requirements formulated in Chapter 3 without additional human assistance.

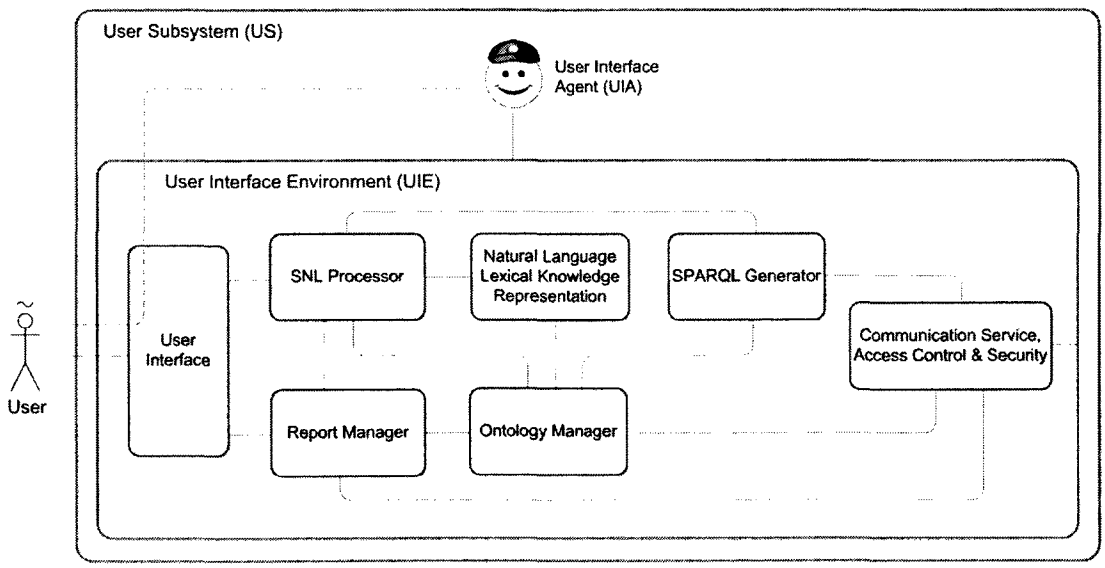

Figure 4.6: The User Subsystem

User Interface Environment

All the components that communicate with the user and the UIA are grouped into the UIE. The solid lines represent direct communication that occurs between the user, the

components and the UIA. The dashed line represents communication that occurs between the user and the UIA. The UIE consists of the following main components:

User Interface

The User Interface (UI) enables all communications between the user and the system. It provides the functionalities with regards to accessing the system as formulated in 4.1.3.

SNL Processor

The SNL Processor component enables the user to interact with the system using the SNL. The user formulates requests for information and modifications to the custom ontology using SNL statements. The SNL processor generates intermediate representations from these statements in three steps. First, it performs a lexical analysis in which it breaks the statements into smaller pieces called *tokens*, which are atomic units of the statements such as words and symbols. Second, it performs a syntax analysis by parsing the token sequence to identify the syntactic structure of the statements. Third, it performs a semantic analysis by verifying the custom and reference ontology to ensure that the tokens are positioned according to their semantic relationships so that each statement as a whole is meaningful. Once successfully generated, the SNL Processor forwards the intermediate representations to the relevant component. If the statements concern a request for information, The SNL Processor forwards the statements regarding what information is to be extracted to the SPARQL Generator, and the statements for formatting the extracted information to Report Manager. Otherwise, it forwards the statements to the Ontology Manager.

SPARQL Generator

The SPARQL Generator constructs a SPARQL script from the intermediate representation of user requests for information received from the SNL Processor. While constructing a script, it refers to the Ontology Manager for the RDB-specific names of terms used in the requests. Once a SPARQL script is generated, the UIA sends it to the DBS for further processing.

Report Manager

The Report Manager presents requested information in the form of reports. It receives SPARQL query results from the DBS and formats the results according to the user's formatting preferences. It communicates with the Ontology Manager to replace any database-specific name in the report with its primary name if the database-specific name is not the primary name. The Report Manager allows the user to view, reformat, save, and delete reports.

Ontology Manager

The Ontology Manager is responsible for maintaining the custom ontology and providing ontological services to the SNL Processor and SPARQL Generator. The custom ontology defines user-specific concepts and their relationships using constructs from the reference ontology. In order to keep the custom ontology in consistency with the reference ontology, any update in the reference ontology needs to be reflected in the custom ontology, if the update affects the definitions of any concepts in the custom ontology. When there is an update in the reference ontology, the UIA checks whether this update affects the custom ontology. If it does, the UIA makes the required changes in the custom ontology. If the update requires simple

operation such as renaming a reference-ontological construct, the UIA performs this action without involving the user. If it requires more complex operations, such as restructuring certain relationships, the UIA engages with the user in the process of updating the custom ontology.

Natural Language Lexical Knowledge Representation

This component provides the meaning and semantic-relations between natural-language concepts in both machine processable and human readable format. In principle, it contains language ontology which can be enhanced by ontological development by a software agent but doing so would be beyond the scope of this thesis. The UIA and the SNL Processor communicates with this component to look up meaning and relationships between natural language terms.

Communication Service, Access Control and Security

This component facilitates communications that occur between the US and the DBS. User privilege and security features are enforced by this component. System access is based on user authentication, which verifies the user's identity and authorization to access specific resources.

User Interface Agent

The UIA interacts with the user through the UIE. The user formulates requests for information and develops a custom ontology through the User Interface. The user interacts with the system using the SNL. The UIA assists the user in the process of formulating requests for information as well as developing and maintaining a custom ontology. The UIA invokes different components in the UIE in order to carry out its tasks. It perceives

the behaviors of the components and through its actions the agent can influence the behavior of the components.

The Database Subsystem

The architectural structure of the DBS is shown in Figure 4.7. The Database Interface Environment (DBIE) comprises the components that provide the main subsystem functions. The primary purpose of the DBIE is to extract requested information from the legacy RDB system, without the need to engage in reasoning in the sense of artificial intelligence techniques. The DBIE components can also be designed and implemented in the same way as the UIE components.

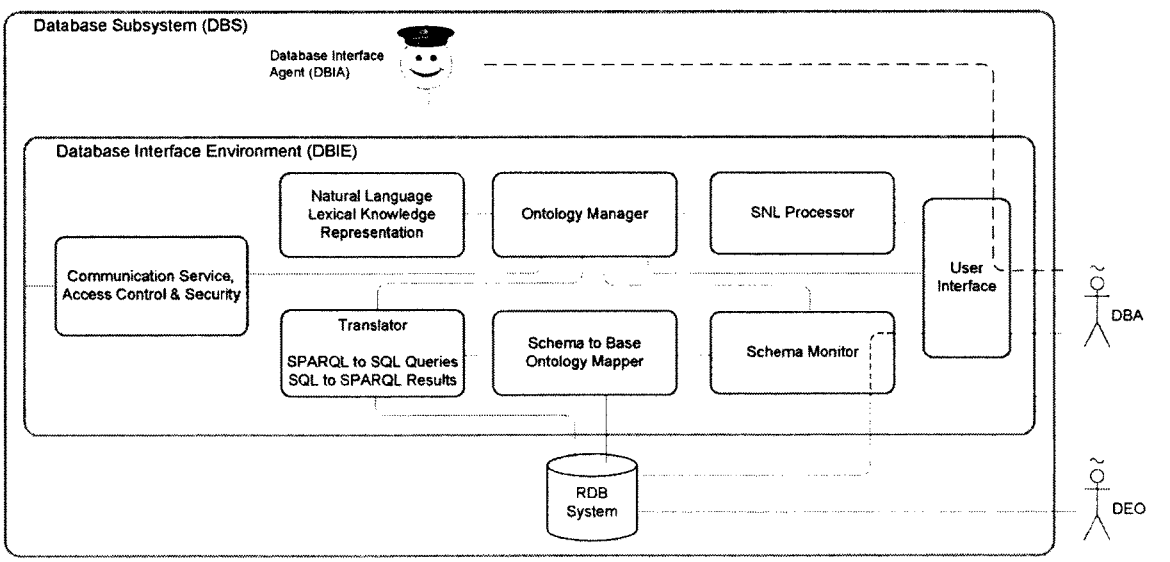

Figure 4.7: The Database Subsystem

Database Interface Environment

All the components that communicate with the DBIA, the DBA, and the DEO are

grouped into the DBIE. The solid lines represent direct communication that occurs between the components. The dashed line represents communication that occurs between the DBA and the DBIA. The DBIE consists of the following main components:

User Interface

The User Interface (UI) provides an access point in which the DBA interacts with the DBS. Through the UI the DBA builds and maintains the reference ontology with the assistance of the DBIA and modifies the structure of the RBD system.

SNL Processor

The SNL Processor component enables the DBA to interact with the system using SNL statements. The DBA interacts with the DBIA in developing and maintaining the reference ontology. The DBA enters SNL statements through the User Interface. The SNL processor then generates an intermediate representation from the DBA's statements in the same three steps the SNL Processor in the US follows. Once the intermediate representation is generated, the SNL Processor forwards it to the DBIA if the interaction concerns developing and maintaining the reference ontology, or to the RDB system if the interaction concerns managing the database.

Ontology Manager

The Ontology Manager is responsible for the coordination and maintenance of the reference ontology. The DBS exports a copy of the reference ontology to the attached US. Thus the reference ontology is replicated in both subsystems. The Ontology Manager ensures that any modifications to the reference ontology in the DBS are propagated to the instances of reference ontology in all participating USs.

Natural Language Lexical Knowledge Representation

This component is identical to *Natural Language Lexical Knowledge Representation* in the US. The DBIA communicates with this component while developing and maintaining the reference ontology.

Translator

The Translator generates SPARQL query results from RDB data in three steps. First, it converts the SPARQL script to SQL queries; second, it executes the SQL queries on the RDB system and retrieves the SQL query results; and finally it converts the SQL query results to SPARQL query results. The DBIA then sends the SPARQL results to the US.

Schema to Base Ontology Mapper

This component automatically generates a base ontology from the underlying RDB schema. The base ontology represents an RDB table name as a class and the column names of the corresponding table as properties of the class. It also captures the relationships between RDB tables. The base ontology serves as a rudimentary ontology from which the reference ontology is incrementally developed.

Schema Monitor

The Schema Monitor always listens for change in the RDB schema made by the DBA. When it detects a schema change it notifies the Schema to Base Ontology Mapper component to reflect the modifications in the reference ontology.

Communication Service, Access Control and Security

This component facilitates all communications between the US and the DBS. By

enforcing security features it ensures that no unauthorized access occurs in the RDB systems.

RDB System

The RDB system contains relational data which the user of SRGS is interested in. The Data Entry Operator (DEO) may insert, delete, or modify data in the RDB system. SRGS is not affected by such modifications.

Database Interface Agent

The Database Interface Agent (DBIA) communicates with the DBA and the DBIE. The DBIA is primarily responsible for assisting the DBA in developing and maintaining the reference ontology. It invokes different components in the DBIE. It perceives the behaviors of the components and through its actions the agent can influence the behavior of the components.

4.2.2 Multiple Users accessing single User Subsystem

In this version of the system architecture, several users may access the US which is attached to only one DBS. The multiplicity of users affects the architecture in the following ways:

1. When there are multiple users accessing the US at the same time, a single UIA is inadequate for attending to all users simultaneously. The preferred solution is to have an agent serving each user. The UIA can dynamically create an agent or awaken up an inactive agent from the background, and assign the agent to the additional user. The architecture of the system is illustrated in Figure 4.8.

2. Some elements in the UI are customized to each user's preferences for interacting with SRGS. Some elements that serve common functionalities to all users remain the same as in the basic configuration of the architecture. The customization of the UI is depicted in Figure 4.9.

3. A custom ontology needs to be created for each user. The Ontology Manager maintains a user's custom ontology by defining the terms that the user may introduce in the custom ontology.

4. The Report Manager maintains each user's history of preferences for report formats.

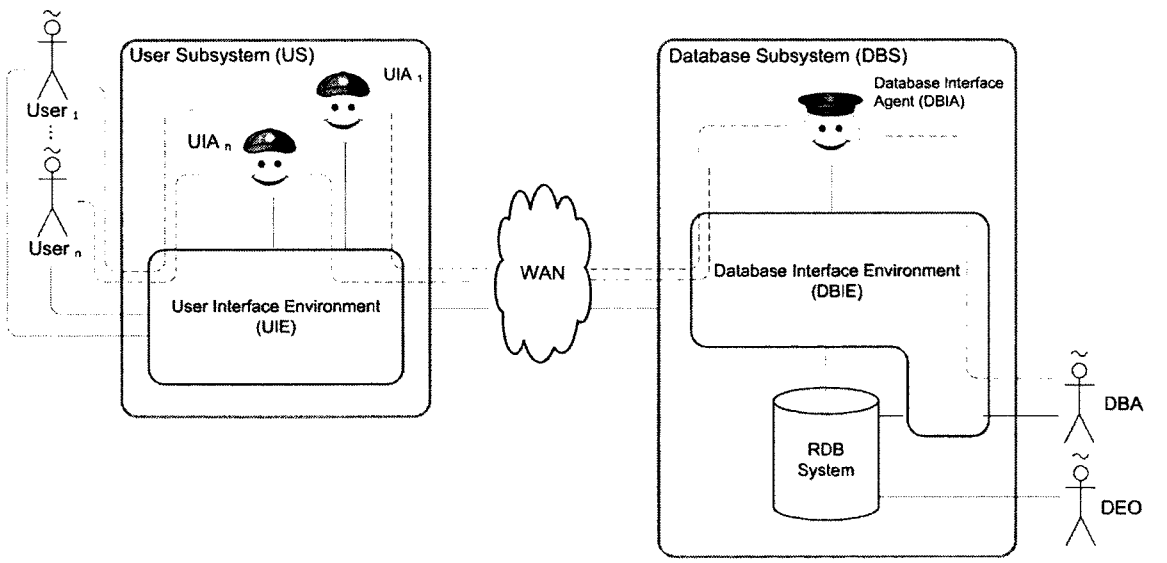

Figure 4.8: Multiple Users accessing SRGS

The DBS is not affected by this change as the DBIA's tasks remain the same as in the basic architecture, regardless of the multiplicity of the users or the USs. The

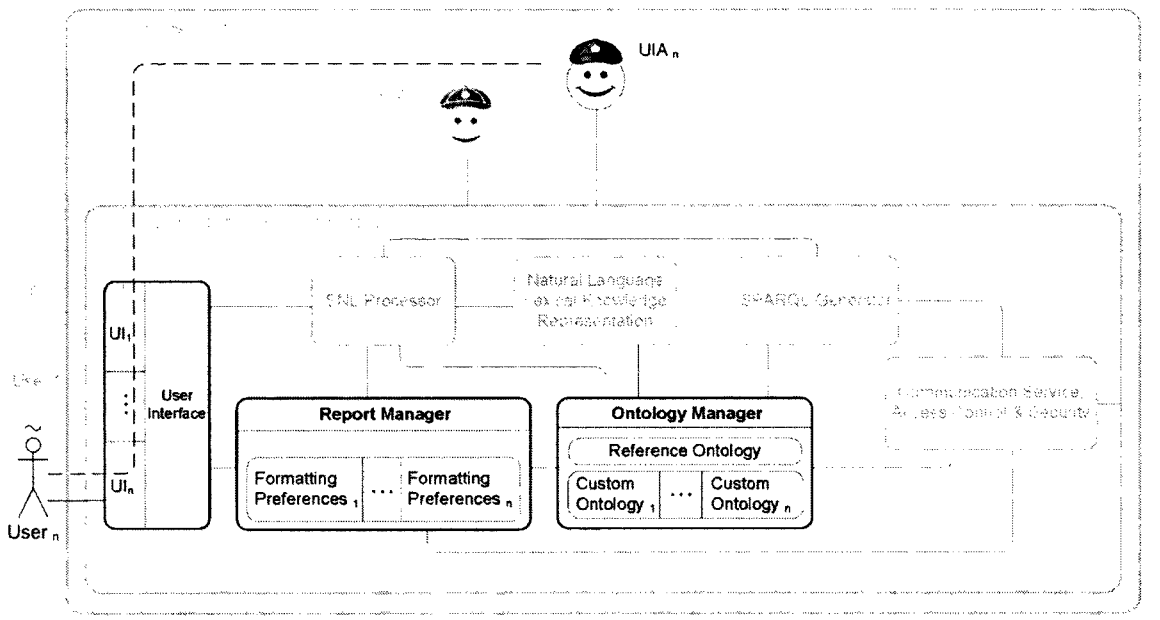

Figure 4.9: Customized User Subsystem for multiple users

DBIA's workload may increase in which case more resources can be added in the actual implementation.

4.2.3 Multiple User Subsystems to single Database Subsystem

In this version of the system architecture, multiple USs interact with a single DBS. This configuration has no effect on the internal design of each US. Implementing this version of the architecture may require certain hardware configurations which are beyond the scope of this thesis.

4.2.4 Single User Subsystem to multiple Database Subsystems

In this version of the system architecture, one US is attached to multiple DBSs. The requirement is that the multiplicity of the DBS is transparent to the user accessing the

US. This transparency can be achieved in the following manner:

1. I assume that there are n DBSs attached to one US. The SPARQL Generator decomposes the SPARQL script into up to n SPARQL scripts. The UIA then sends each SPARQL script to its respective DBS. Once the scripts are processed, the UIA receives a set of SPARQL results from the DBSs and forwards them to the Report Manager. Note that the UIA must receive SPARQL query results from all the DBSs. The Report Manager aggregates the set of SPARQL results into one resultset. This scenario is illustrated in Figure 4.10.

2. The reference ontology in the UIE is now a union of n reference ontologies, where n is the number of DBSs. The Ontology Manger updates the reference ontology whenever the reference ontology in the DBS is changed. The modified US is illustrated in Figure 4.11.

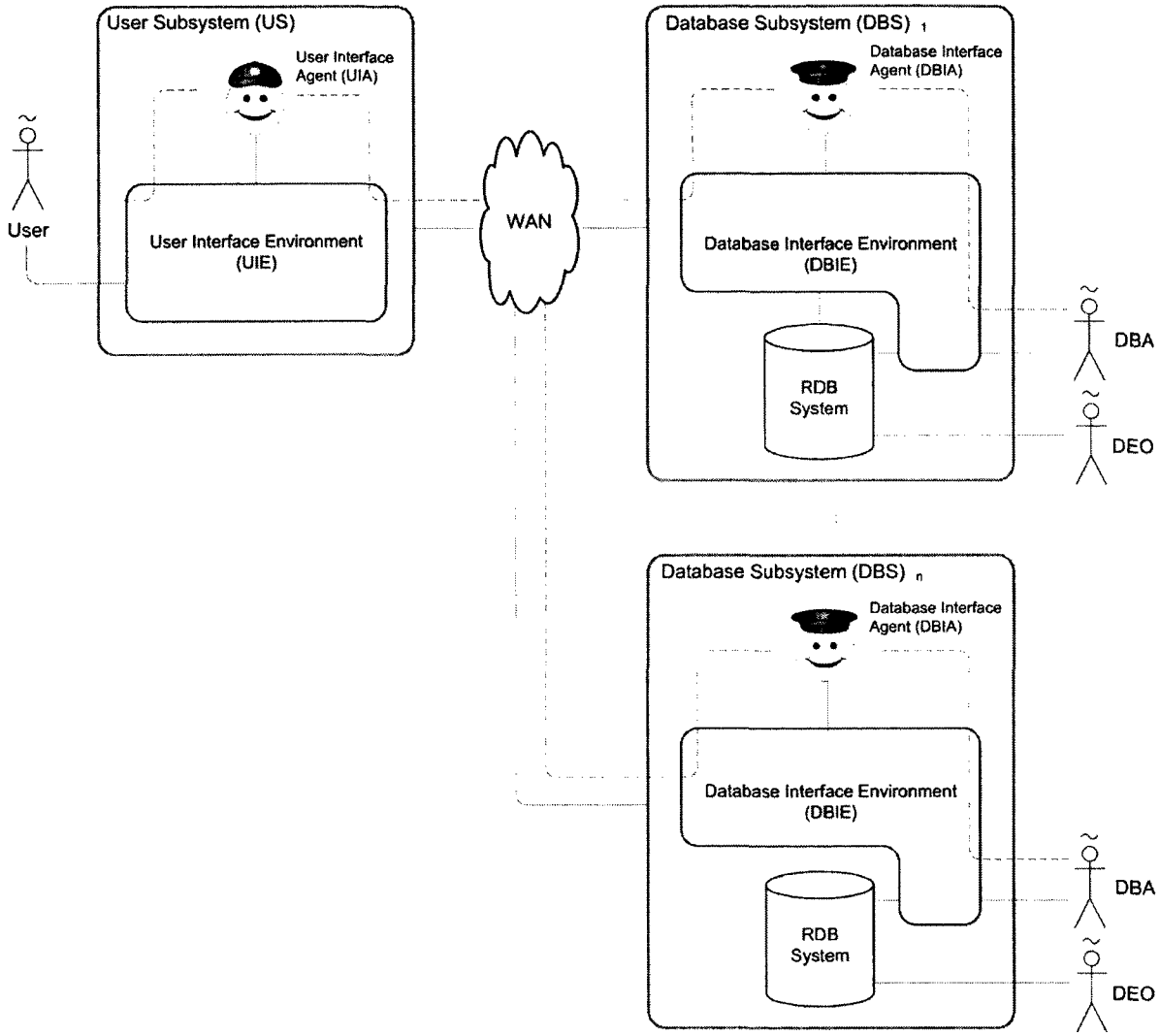

Figure 4.10: Single User Subsystem attached to multiple Database Subsystems

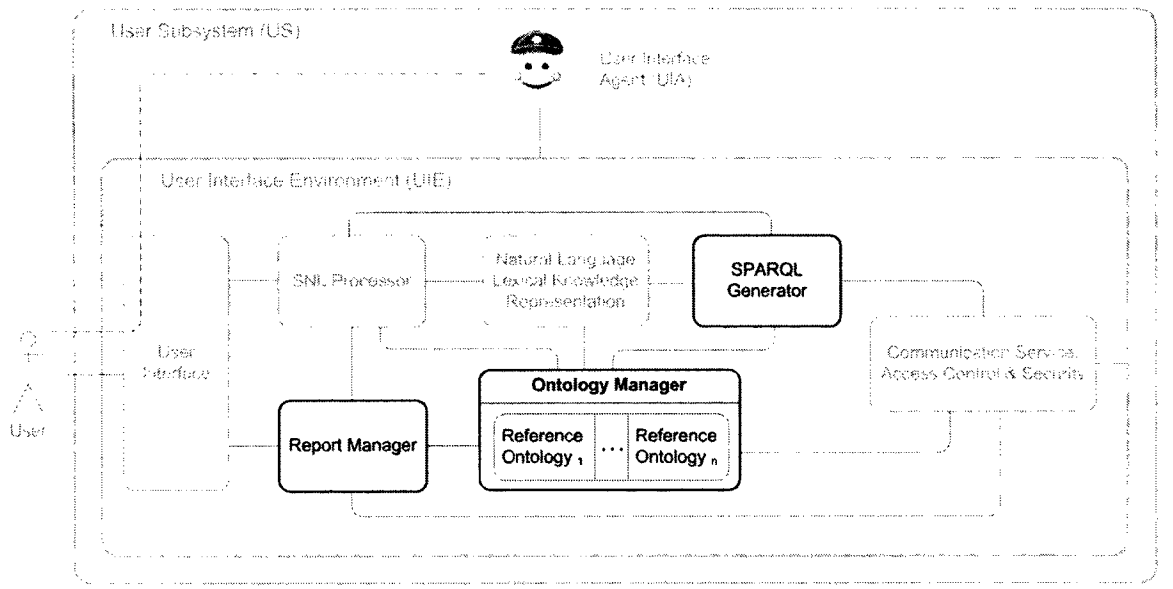

Figure 4.11: Customized User Subsystem for multiple Database Subsystems

4.2.5 Multiple User Subsystems to multiple Database Subsystems

In this version of the system architecture, several users access the US, which is attached to more than one DBSs. It is a combination of the scenarios discussed in Subsection 4.2.2 and Subsection 4.2.4. Several components in the UIE need to be changed in order to accommodate the multiplicity of both subsystems. The User Interface is customized for each user. The Report Manager maintains each user's history of preferences of report formats. The Ontology Manger maintains a custom ontology for each user. The reference ontology is a union of n number of ontologies, where n is the number of DBSs.

4.3 Agent roles

This section provides the specific roles performed by the UIA and the DBIA in SRGS.

User Interface Agent

1. *Assistance in SNL dialogue*

   The UIA assists the user in formulating requests for information and developing a custom ontology. The user interacts with the system using SNL statements. The SNL processor generates an intermediate representation from these statements in three steps. In each step, the SNL processor may produce a warning or an error when the statements are formulated incorrectly. If the SNL processor produces a warning, the UIA perceives this warning and reconciles with the SNL Processor to resolve any arising issues in the statements. If the SNL processor produces an error, the UIA engages with the user to correct the error in the statements.

2. *Searching the Semantic Web*

   The agent can search the Semantic Web to look for two things. First, it can search for language ontologies such as WordNet to look up synonyms and hypernyms of terms. Second, the agent can search for domain ontologies to include terms that are acceptable in wider context.

3. *Development of custom ontology*

   The UIA assists the user in developing a custom ontology, which describes the conceptual framework specific to a the user that can be directly translated to the reference ontology. In order to keep the custom ontology in consistency with the reference ontology, the UIA ensures that any update in the reference ontology is also

reflected in the custom ontology if the update affects any definitions of concepts and their relationships in the custom ontology. The UIA has the technical knowledge of the ontology development process. In addition, the UIA refers to publicly available ontological resources on the Semantic Web.

4. *Customizing the behavior of User Interface*

   While assisting the user with SNL dialog, the UIA can observe if the user is typically making a certain types of choice, and offer this choice first in its next interaction. This choice can be an explicit one in which the user specifies certain preferences or an implicit one in which the agent continuously learns by observing the user's course of actions.

5. *Coordination of reference ontologies*

   There may be multiple reference ontologies if there are multiple DBSs attached to the US. The UIA needs to resolve any conflicting situations that may arise due to the presence of multiple reference ontologies in the US. There can be two terms that are identical across two different ontologies but they may have completely different meaning in their respective contexts. While developing the custom ontology, the user needs to conveniently see these differences. The UIA disambiguates the terms that are seemingly identical but have different meaning. The UIA does this by assigning unique tags to the constructs when they are retrieved from different reference ontologies.

Database Interface Agent

1. *Assistance in SNL dialogue*

   The DBIA assists the DBA to interact with the system while developing and maintaining the reference ontology. The DBA interacts with the system using SNL statements. The SNL processor generates an intermediate representation from these statements in three steps. In each step, the SNL processor may produce a warning or an error when the statements are formulated incorrectly. If the SNL processor produces a warning, the DBIA perceives this warning and reconciles with the SNL Processor to resolve any arising issues in the statements. If the SNL processor generates an error, the DBIA engages with the DBIA to rectify the error in the statements.

2. *Searching the Semantic Web*

   The DBIA can search the Web to look for two things. First, it can search for language ontologies such as WordNet (Miller, 1995) to lookup synonyms and hypernyms of terms. Second, it can search for domain ontologies to include terms that are acceptable in wider context.

3. *Development of reference ontology*

   The DBIA interacts with the DBA in developing and maintaining a reference ontology. The *Schema to Base Ontology Mapper* component analyzes the RDB schema and generates a Mapping File, which contains RDF models of the RDB schema. The Mapping File then serves as a base ontology from which the DBA incrementally builds a full reference ontology with the assistance of the agent. The DBIA is

equipped with the technical knowledge of ontology development process. In addition, it refers to ontological resources on the Semantic Web.

4. *Customizing the behavior of User Interface*

   While assisting the DBA with SNL dialogue, the DBIA can observe if the DBA is typically making a certain types of choice, and offer this choice first in its next interaction. This choice can be an explicit one in which the DBA specifies certain preferences or an implicit one in which the agent continuously learn by observing the DBAs course of actions.

In a multiagent environment, several instances of the same role can be assigned to multiple agents. It is also possible to assign several instances of different roles to the same agent. Some of the roles described above are demonstrated in the use case scenarios presented in Chapter 5.

4.4 Incorporation of existing system components

The proposed architecture relies on existing software tools that have been developed as results of research in the Semantic Web and MAS. This subsection presents the main Semantic Web and MAS tools that can be used as components in the proposed architecture.

The D2RQ Platform (Bizer and Seaborne, 2004) can provide the functionalities of the *Translator*, *Schema to Base Ontology Mapper*, and Schema Monitor components in SRGS. Its front-end, the D2R Server, accepts SPARQL queries and presents SPARQL results (RDB triples). The D2RQ Engine is the core of the platform which provides the conversion service. It analyzes the structure of the RDB and generates a Mapping

File which is RDF representation of the RDB schema. In SRGS, the Mapping File is considered as the base ontology. The Mapping File is further described in A.1. D2RQ Engine uses the Mapping File to convert RDB data to RDF triples.

The D2RQ Platform is an on-demand mapping tool, which dynamically translates RDB data to RDF triples instead of completely transforming an RDB to an RDF triple-store. If a data value in the database is changed, the D2RQ Platform can instantly display the new value. However, when a column or a table in the RDB is altered or dropped it does not display the new value. In other words, the D2RQ Platform fails to detect any change in the RDB schema. In order to overcome this limitation, I have added an extension to the D2RQ Platform. The extension enables the D2RQ Engine to detect any RDB schema change and display the modified values. The extension is further discussed in Appendix A.2. The three DBS components that provide the functions of the D2RQ Platform are shown in the dashed box in Figure 4.12.

JADE (Bellifemine et al., 2007) is an agent-oriented middleware that provides domain-independent infrastructure for developing multiagent systems. JADE complies with the Foundation for Intelligent Physical Agents (FIPA) specifications and includes a set of tools that supports debugging and deployment tasks. The agent platform can be distributed across multiple computers and the configuration can be controlled via a remote Graphical User Interface. The configuration can be changed at run-time by creating new agents.

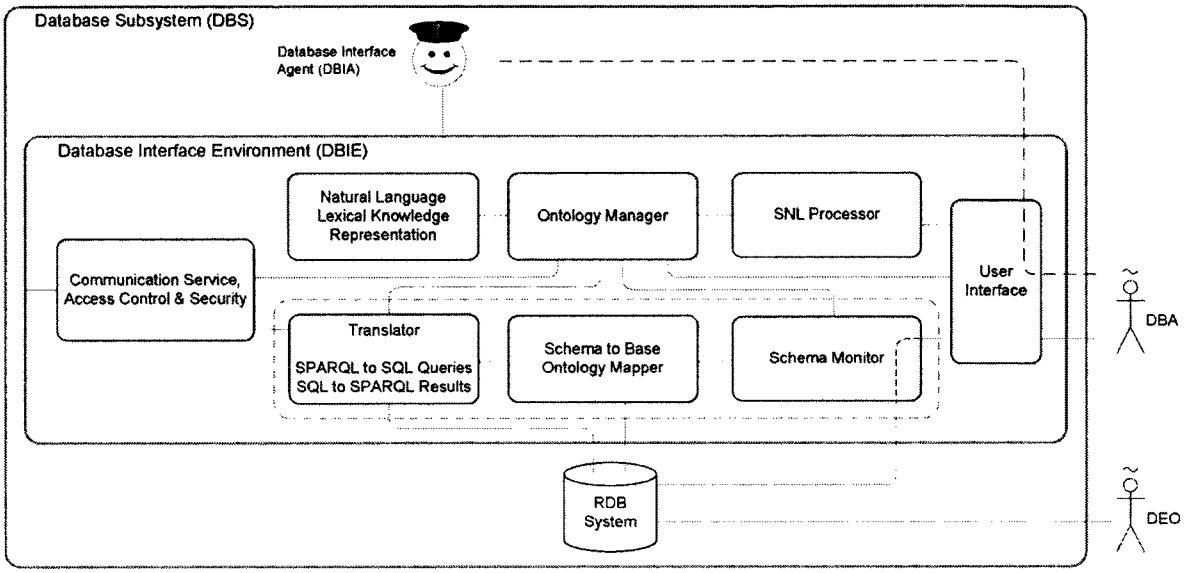

Figure 4.12: The role of D2RQ Platform in the DBS

In order for Jade agents to work with the D2RQ Platform, a glue component is required. Jena (McBride et al., 2010), a Java based framework for extracting data from RDF triplestore, can function as an interfacing component between Jade and the D2RQ Platform. A JADE agent uses Jena's SPARQL capabilities for executing a SPARQL query on the D2RQ Platform.

Chapter 5

Modeling and Accessing Information in SRGS

In this chapter, I present three scenarios that illustrate the behavior of the proposed architecture of the Semantic Report Generation System (SRGS). Each scenario demonstrates a specific behavioral aspect of SRGS. The first scenario shows how the user can access information stored in the RDB system that underlies SRGS. The other two scenarios show the activities involved in developing the SRGS ontologies from the schema of the RDB.

In the scenarios, I personalize the actors and the software agents by giving them human-like names: the user is called *Henry*, the User Interface Agent (UIA) is called *Alice*, the Database Administrator (DBA) is called *Helen*, and the Database Interface Agent (DBIA) is called *Adam*. For consistency, the human actors' names start with the letter ***H***, and the agents' names start with ***A***. The remainder of this chapter is organized as follows: agent oriented ontology development is discussed in Section 5.1; a scenario

on accessing information using SRGS is presented in 5.2; a scenario on developing a reference ontology from an RDB schema is given in 5.3; and a scenario on developing a custom ontology, which defines user-specific concepts using constructs from the reference ontology, is shown in 5.4.

5.1 Ontology development by software agents

In this section, I analyze the roles of the SRGS agents as ontology builders. The notion of ontology development by software agents is not mentioned in the literature I have reviewed; it is a novel aspect of this approach. In this approach, the agents interact with human actors throughout the entire ontology development process. The agents perform some of the technical tasks and make suggestions, while the human actors make decisions. This human actor role in ontology development adds a new dimension to the traditional user and DBA profiles. However, this does not require them to become technical experts fully specialized in the ontology development process because the agents are responsible for executing some of the technical tasks.

The SRGS agents must have the requisite knowledge of how to build an ontology in order to fulfill their role as ontology builders. That knowledge itself is formally represented as an ontology (for the ontology-building knowledge domain), to which we refer as *meta-ontology* (note that this usage of the term differs from its established meaning in the realm of philosophy and metaphysics). The meta-ontological knowledge includes an understanding of the semantics of general ontological notions, such as class and relationship. The agents rely on their meta-ontology to provide technical guidance to their human partners in the construction of concrete ontologies for the specific knowledge do-

main to which the relational database belongs.

An ontology language is required to formally define classes, properties and relationships in the ontology. The Web Ontology Language (OWL) provides the necessary primitives for defining concepts and their relationships. OWL has three sublanguages: *OWL Lite*, *OWL DL*, and *OWL Full*. OWL Lite, which supports a classification hierarchy and simple constraints, is intended for building ontology from a thesauri or a taxonomy of terms; OWL DL provides maximum expressiveness with certain restrictions while retaining computational completeness and decidability, which means that all conclusions are guaranteed to be computable in finite time; OWL Full provides maximum expressiveness and the syntactic freedom of RDF (for instance, a class can be treated as an instance of another class) but does not guarantee computational completeness, therefore, a reasoning software may not be able to support complete reasoning for every feature of OWL Full (McGuinness and Harmelen, 2004). In this thesis, I use OWL DL as I am more familiar with it, and it provides all the required primitives to develop SRGS ontologies. Section 2.4 summarizes the main OWL DL primitives. The scenarios presented in the next sections are accompanied with relevant OWL statements to illustrate how agents can use these primitives while building an ontology.

OWL is based on the Open World Assumption (OWA). This assumption states that information presented in a knowledge base is not complete; therefore, when a statement cannot be directly proved from the asserted facts and rules of inference, the reasoner cannot infer that thereby the statement is false. OWA is suitable for information presented on the Web because one cannot assume its completeness. On the other hand, an

information system of an enterprise or a university is considered complete, and reasoning within it should be based on the Closed World Assumption (CWA). Within SRGS, the reasoning involving the information stored in the RDB is based on CWA; therefore, any statement that is not explicitly provable to be true is considered false. (However, the SRGS agents use OWA when reasoning about information contained in external knowledge sources on the Semantic Web.) Using OWL to develop an ontology from an RDB schema thus presents a contradiction: OWL uses OWA whereas CWA is more appropriate for the RDB. There are suggestions (Tao et al., 2010; Horrocks et al., 2005) towards resolving this issue, but no standard mechanism has been recommended by the W3C.

There is another contradiction concerning the use of OWL for reasoning within SRGS. The CWA is often complemented by the Unique Name Assumption (UNA), which states that names in a knowledgebase are unique and refer to distinct instances. OWL does not make UNA since there can be different names denoting the same instance. However, it provides several mechanisms to address the issue of non-uniqueness. The common solution is to explicitly state using the "`owl:sameAs`" primitive that two Uniform Resource Identifier (URI) references refer to the same instance. The uniqueness of the two URI references satisfies UNA in an ontology.

In SRGS, there can be multiple names describing the same instance of a given concept. Among them, a *primary name* is designated by the human actor as the official name of the concept instance in the reference ontology; each concept instance derived from the RDB schema also has a *base name*, which may or may not be the same as the primary name. All possible names of the same concept instance are semantically linked to its pri-

mary name. The system tolerates the usage of the other possible names by human actors, but the agents refer to the primary name when reasoning and interacting with the human actors. One drawback of OWL when it comes to implementing such a naming convention is that it does not provide a mechanism to establish a primary name in the ontology. As OWL continues to mature I believe this is one aspect where further research can be done.

In order to realize the proposed naming convention for SRGS, the following mechanism is adopted. When defining the name of a concept instance in the ontology the SRGS agents append to the name a leading tag followed by a period (.) which works as a separator between the tag and the name. The tags "bn", "pn", and "un" are used for each base name, primary name, and user-specific name respectively. The leading tag allows the SRGS agents to identify the type of the name of a concept instance. The agents remove the tag from a name before displaying it to the human actors. OWL also does not support blank space in names. In SRGS, the agents insert a hyphen (-) to replace any blank space appearing in names entered by the user or retrieved from external lexical dictionaries. The agents remove the hyphen (-) from a name and insert a blank space when displaying the name to the user.

In this thesis, I use OWL for illustrative purposes only, and ignore its limitations discussed above. OWL was conceived with the goal of building ontologies for the Semantic Web. Due to its lack of CWA and UNA, OWL may not be the ideal choice for modeling ontology developed from an RDB schema. OntoDLP (Ricca et al., 2009), which is based on Answer Set Programming, is more suitable for representing an RDB schema as ontology since it complies with both CWA and UNA. I choose OWL as it is the commonly

known standard ontology language for the Semantic Web. The research question of finding or developing an ontology language that is best suited to the requirements of SRGS is beyond the scope of this thesis.

In SRGS, agents reason with their own meta-ontologies, ontologies developed within SRGS, and public ontologies on the Semantic Web. The agents adopt the CWA when reasoning with ontologies that are within the boundary of SRGS, and the OWA when reasoning with ontological constructs imported from external knowledge resources. A detailed description on reasoning from imported ontological constructs is given in Ricca et al. (2009).

The SRGS agents are also equipped with the know-how of specific steps involved in the ontology construction. In the scenarios that follow, the methodological steps in developing ontology (discussed in Subsection 2.3.2) have been adapted for SRGS. The human partners make decisions and also need a good understanding of the ontology building process, but owing to the agent guidance need not have the level of expertise of specialist ontology developers.

In SRGS, an agent in interaction with a human partner builds a reference ontology from an RDB schema. In this process, the D2RQ Platform analyzes the RDB schema and generates a Mapping File, which contains serialized RDF models of the RDB schema. It is written in the D2RQ mapping language — a declarative language for describing the relation between an ontology and an RDB schema. The Mapping File serves as a base ontology from which the agent, in interaction with the human partner incrementally

builds a full reference ontology following the adapted know-how of ontology development steps.

The agent has the capability to understand the syntax of the Mapping File. It extracts the names of classes and properties from the Mapping File, and uses them to define the corresponding base classes and properties in the reference ontology. It then interacts with the human partner to extend the ontology with more general classes, and to introduce the properties with meaningful names. The agent also extracts from the Mapping File the relations between classes, and with the help of the human partner defines the corresponding class-relations in the reference ontology. Every concept in the reference ontology must ultimately be reducible to the concepts of the base ontology; otherwise, high-level queries using reference ontology may not be translatable to SQL queries. The Mapping File is further elaborated in Appendix A.1

A custom ontology is developed using relevant constructs from the reference ontology. In this process, another agent assists the user to introduce user-specific concepts in the custom ontology.

5.2 Scenario 1: Accessing information in SRGS

This scenario shows how the user (Henry) can request for information stored in the RDB through the User Subsystem (US). The starting assumption is that the US has a copy of the reference ontology, which is consistent with the reference ontology in the DBS. The US also has a custom ontology, which defines user specific concepts using constructs in the reference ontology. Both ontologies are used in processing Henry's request.

Henry formulates a request for report in the Simplified Natural Language (SNL) and submits it through the User Interface (UI). An example of the SNL request is shown in Figure 5.1.

```
Generate list of students that registered for Fall 2011.
Include in the list student ID, first name, last name, date of birth, CGPA.
Format report using format-k with title: Registered Students;
subtitle: Date: today's date; sort list alphabetically by Last Name.
```

Figure 5.1: The SNL request for report

Some assumptions are made about the SNL request. There are three categories of words in the request. The first category consists of control structure of the language based on simplified English; the second category consists of commands that specify invocation of actions by system components; and the third category comprises the words that have technical meaning in the custom and reference ontologies. In all three categories user has the flexibility of defining custom terms. In this scenario, we only consider the translation of custom ontology terms. A request has three statements. The first statement tells what information is to be retrieved; the second statement gives additional details as to what specific information is to be included in the report; and the third statement describes how the information is to be formatted.

The SNL Processor generates an intermediate representation of the request in three steps. In the first step, it performs lexical analysis in which it breaks the SNL text into small pieces called *tokens*, which are atomic units of the request, such as words and symbols. In the second step, the SNL Processor performs syntactic analysis to ensure that

the grammar of the SNL text is correct. In the third step, it does semantic analysis to verify the semantic correctness of the statements according to the custom and reference ontologies. During this step, it recognizes the actions and verifies whether all parameters that are needed for these actions are present. It also creates intermediate representation where every ontological term is brought to the base name.

In each of these three steps, the SNL processor may produce a warning or an error. For example, in lexical analysis, the SNL processor searches the custom and reference ontology to determine whether a word has a technical meaning if it is found in the ontologies. A word may be found to have both general meaning in the SNL Processor's vocabulary, as well as technical meaning according to the ontologies. The SNL Processor flags this word as a warning. The User Interface Agent (Alice) perceives this warning and reconciles with the SNL Processor. In such case, the technical meaning takes precedence over the general meaning. For example, the words *First* and *Name* both have general meaning in English, but together *First Name* is found as a property of the class *Student* in the reference ontology, therefore, *First Name* has a technical meaning. Alice does not display this warning to Henry. She instructs the SNL Processor to accept *First Name* as a technical term instead of general words.

The SNL processor also flags an error when a word has neither general nor technical meaning. Alice perceives such error messages and analyzes them by consulting a lexical dictionary. Using the word in the error message, Alice searches the lexical dictionary for matching words and displays them to Henry with an explanation that a word in the request does not have any general or technical meaning. Henry selects the proper word

and Alice passes the word to the SNL Processor.

For the request in Figure 5.1, the SNL Processor finds matching constructs for each word except *registered*. It produces the following error message: *'registered' not found in the ontology*. Alice perceives this error message and using the word *registered* searches the lexical dictionary for synonyms. Alice compares every found words with the primary or user-specific names of ontology constructs and if a match is found, asks Henry for approval. If no match is found, Alice displays the primary and user-specific names of classes that have the property name *StudentID* and uses the selected name for further processing. In this example, *Registration* is the class name that is presented to Henry by Alice and approved by Henry.

The SNL Processor checks whether each technical word is a base name in the reference ontology. Otherwise, it retrieves the corresponding base name from the reference ontology and substitutes the technical word with the corresponding base name in the generated intermediate representation. For instance, the SNL Processor finds *DOB* is the base name for *Date of Birth* in the reference ontology, hence it replaces *Date of Birth* with *DOB* in the intermediate representation. It also determines what type of construct (i.e. class, property, etc.) a technical word is in the ontologies and marks the word with its construct type. This later helps the SPARQL Generator in constructing the SPARQL script.

The SNL Processor processes its own command words such as *Generate*, *Include*, and *Format*, as well as the parameters that are associated with these commands. After processing these commands, the SNL Processor forwards the first two statements to SPARQL Generator and the third statement to Report Manager. The first statement specifies the criteria for extracting information from the database, and the second statement gives additional details as to what specific information is to be included in the report. The SPARQL Generator uses these two statements to construct the SPARQL script. The third statement states in which way the extracted information has to be formatted. The Report manager uses the third statement in formatting the extracted information as report.

The SPARQL Generator distinguishes the technical words in the first statement into two categories: basic-technical-words set with words appearing before the keyword *that* and conditional-technical-words set with words that follow. This distinction later helps the generator in constructing the script.

The SPARQL Generator constructs a SPARQL script from the commands and parameters received from the SNL Processor. The script may be created with more than one SPARQL queries. In this scenario, I show one SPARQL query. A SPARQL query is made up of three components: PREFIX declaration, SELECT clause, and WHERE clause. The generator gets the base URI `base="http://localhost:2020/vocab/resource/"` from the reference ontology header and includes in the query as a PREFIX. In the prefix declaration, it replaces the equals symbol (=) with a colon (:), and the quotes with an opening (<) and closing (>) tag.

It then constructs the body of the query consisting of a SELECT clause and a WHERE clause. The SELECT clause identifies the variables to appear in the query results. In the SELECT clause, variables are taken from the technical words appearing in the second statement of the SNL request. The generator appends a leading "?" symbol to each base name to make it a variable. In the example SPARQL query shown in Figure 5.2, the variables in the SELECT clause are *?StudentID*, *?FirstName*, *?LastName*, *?DOB*, and *?CGPA*.

In the WHERE clause, a number of triple patterns are constructed. A triple pattern consists of a subject, a predicate, and an object. The subject is a variable created by appending the "?" symbol to the class name from the basic-technical-words set. The predicate is a technical word written in the URI format (`PREFIX:Class_Property`), which is constructed in the following two steps: First, The SPARQL Generator concatenates a class name and a property name with an underscore symbol (_) in between them. The class name comes from the basic-technical-words set and the property name comes from the SELECT clause. Second, it concatenates the prefix (`base`) and the previously created segment (`Class_Property`) with a colon symbol (:) in between them. The object variable is constructed using the property name. Following this method the generator constructs a triple pattern for each variable appearing in the SELECT clause.

The generator then constructs a triple pattern for each property name from conditional-technical-words set. This time it uses the words from the conditional-technical-words set to create the variables. These two groups of triple patterns are then linked with a third triple pattern whose predicate has the property *StudentID*, which is a common prop-

erty between the class in the basic-technical-words set and the class in the conditional-technical-words set. The SPARQL script in Figure 5.2 is constructed from the example SNL request. Once the SPARQL script is constructed, the Communication Service, Access Control & Security (CSACS) sends it to the destination DBS.

```
PREFIX base: <http://localhost:2020/base/resource/>
SELECT ?StudentID ?FirstName ?LastName ?DOB? ?CGPA
WHERE {
   ?student a vocab:Student.
   ?registration a vocab:Registration.
   ?student base:Student_StudentID ?studentID.
   ?student base:Student_FirstName ?FirstName.
   ?student base:Student_LastName ?LastName.
   ?student base:Student_DOB ?DOB.
   ?student base:Student_CGPA ?CGPA.
   ?registration base:Registration_StudentID ?student.
   ?registration base:Registration_Semester "Fall".
   ?registration base:Registration_Year "2011".
}
```

Figure 5.2: The SPARQL script

The CSACS in the DBS receives the SPARQL script. By verifying credentials of the sender, it ensures that no unauthorized access occurs to the RDB system. It then passes the SPARQL script to the Translator component, which decomposes the script into one or more SPARQL queries. The D2RQ Engine within the Translator generates equivalent SQL queries by rewriting the SPARQL queries to RDB-specific SQL queries. The SQL query shown in Figure 5.3 is generated from the SPARQL query in Figure 5.2.

```
SELECT Student.StudentID, Student.FirstName,
       Student.LastName, Student.DOB, Student.CGPA
FROM Student, Registration
WHERE Student.StudentID = Registration.StudentID
      AND Registration.Semester = 'Fall'
      AND Registration.Year = '2009'
```

Figure 5.3: The SQL query

D2RQ query engine executes the SQL queries on the RDB system and retrieves SQL results. The Translator then converts the retrieved results from SQL format to SPARQL format. Note that the SQL results and the SPARQL results are the same specific information retrieved from the database. For compatibility reason the Translator converts the retrieved results from SQL to SPARQL format. Once the SPARQL results are generated, the CSACS sends them to the US. A subset of the generated SPARQL results is shown in Figure 5.4.

StudentID	FirstName	LastName	DOB	CGPA
98988	Shen	Ming	1988-12-22	3.25
44553	Phill	Cody	1990-05-10	3.7
98765	Emily	Brandt	1978-10-29	2.85
70665	Jie	Zhang	1990-08-26	3.4
76543	Lisa	Brown	1992-06-01	3.7
19991	Shankar	Patel	1986-02-17	3.65
70557	Amanda	Snow	1989-01-17	3.1
76653	Tom	Anderson	1984-03-20	3.5

...

Figure 5.4: The SPARQL results

The Report Manager in the US receives the SPARQL results from the DBS. It then formats the results according to the formatting instructions provided in the request by Henry. It adds the report title *Registered Students* and the subtitle *Date* for the report generation date. A user selected template (format-k) is used for displaying the report. It also sorts the SPARQL results alphabetically by *LastName*. The Report Manager refers to the Ontology Manager to replace any base name with its primary name or user-specific name. It then displays the formatted report to Henry. The report generated from the SPARQL results is shown in Figure 5.5.

Registered Students

Date: September 15, 2011

Student ID	First Name	Last Name	Date Of Birth	CGPA
76653	Tom	Anderson	1984-03-20	3.5
98765	Emily	Brandt	1978-10-29	2.85
76543	Lisa	Brown	1992-06-01	3.7
44553	Phill	Cody	1990-05-10	3.7
98988	Shen	Ming	1988-12-22	3.25
19991	Shankar	Patel	1986-02-17	3.65
70557	Amanda	Snow	1989-01-17	3.1
70665	Jie	Zhang	1990-08-26	3.4
...				

Figure 5.5: The formatted report

5.3 Scenario 2: Developing reference ontology

This scenario presents the specific steps involved, and the interactions that occur between the Database Interface Agent (Adam) and the Database Administrator (Helen), in developing a reference ontology from an RDB schema. The steps are presented in the following order: determining ontology domain and scope, defining classes and class hierarchies, defining properties of classes, and defining relations between classes.

The construction of the reference ontology begins with determining its domain name that accurately reflects its scope. Adam extracts the name of the database from the Mapping File and displays to Helen, asking whether it represents the domain of the application. Helen approves either by accepting the displayed name or entering a different name. Adam then defines the approved name as the domain of the reference ontology. In general there can be several relevant names to describe the ontology domain, but in this case we show one.

Once the ontology domain name has been established, Adam uses it to search for publicly available ontologies in the same domain on the Semantic Web. For example, if the application domain is *university*, Adam searches for university ontology on the Semantic Web. With Helen's approval Adam may include the URI reference link to such an ontology in the reference ontology header. This allows the agent to later selectively import certain constructs from the external university ontology with Helen's approval. Similarly, Helen can specify other relevant domain for which external knowledge bases may be helpful; for instance, importing an external ontology of calendar structures or time zones may be preferable to developing one's own. In the context of developed Se-

mantic Web, review of external ontologies can play a significant role in constructing one's own reference ontology. In the current scenario this process is illustrated only through the use of the external ontology of the English language (such as WordNet (Miller, 1995)).

Once the domain is determined, Adam asks Helen to provide any general comments about the ontology being developed and includes them in the reference ontology. Adam also extracts the prefix statements from the Mapping File (Figure 5.6a) and defines them as Extensible Markup Language (XML) namespaces above the reference ontology header (Figure 5.6b). Using the prefix `vocab: <http://localhost:2020/vocab/resource/ >`, Adam defines the base namespace `xmlns:base = "http://localhost:2020/vocab/ resource/"`, which provides a means to unambiguously identify constructs in the reference ontology from constructs in an imported ontology (which come with their own base namespace prefix). The remaining namespace definitions enable writing names in shorter forms, such as `rdf` instead of `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.

The class names and their synonyms are defined next. The set of class names is formed in two steps: first, the names of *base classes* are extracted from the base ontology represented by the Mapping File; second, the names of higher level classes are introduced in interaction between Adam and Helen. In the first step, Adam extracts base class definition entries, such as `map:Student a d2rq:ClassMap;`, from the Mapping File and defines corresponding classes in the reference ontology. Since the base class names are RDB table names, they may not always be sufficiently descriptive for user level communication. Therefore, Adam presents each base class to Helen, who can respond in three ways. First, she may decide that the base class name can adequately serve as the

```
@prefix vocab: <http://localhost:2020/vocab/resource/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

map:database a d2rq:Database;
   d2rq:jdbcDriver "com.mysql.jdbc.Driver";
   d2rq:jdbcDSN "jdbc:mysql://localhost/University";
```

(a) D2RQ Mapping

```
<rdf:RDF
   xmlns:base = "http://localhost:2020/vocab/resource/"
   xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema#"

<owl:Ontology rdf:about="University">
   <rdfs:comment>
      Reference ontology developed from
      the university RDB schema
   </rdfs:comment>
   <owl:versionInfo> V1.1 2011/09/15 </owl:versionInfo>
   <owl:imports rdf:resource=""/>
   <rdfs:label> University Ontology </rdfs:label>
</owl:Ontology>
```

(b) OWL

Figure 5.6: (a) Prefix and RDB details in Mapping File (b) XML namespaces and ontology header in reference ontology

primary class name and approve it as such; second, Helen may provide an alternative primary name and approve it immediately; third, Helen may provide a tentative choice of primary name, asking Adam to conduct an external synonym search, and decide which primary name to approve after reviewing the synonym choices. (At this point Helen does not have the option of introducing other synonyms for the class name, even though class name synonyms in the reference ontology are allowed their use is restricted to the maintenance of backward compatibility between ontology versions.) Figure 5.7 illustrates the definition of the class *Department* in (a) the Mapping File and in (b) the reference ontology.

```
map:Department a d2rq:ClassMap;
```

(a) D2RQ Mapping

```
<owl:Class rdf:ID="bn.Department"/>
```

(b) OWL

Figure 5.7: Class definition in: (a) Mapping File and (b) reference ontology

Once the base classes are defined, Adam and Helen define the more general classes. The superclasses can be defined in four ways. First, Helen identifies several existing classes that can be generalized into a new superclass; she provides the primary name for the superclass and Adam creates it. Second, Helen identifies the existing classes and provides a tentative primary name for the superclass; Adam then searches for synonyms of that name in external lexical ontologies, and Helen approves the new class names after reviewing the choices. Third, Helen identifies the existing classes and asks Adam to suggest possible superclass names; Adam searches for hypernyms of each existing class name and reports to Helen the intersection of the hypernym sets resulting from the searches; Helen then approves the primary name of the new superclass after reviewing the choices. Fourth, Adam finds the hypernym set of each existing class, forms all possible intersections, and reports to Helen each nonempty intersection; in each case, Helen decides whether a new superclass is needed and, if so, approves its primary name after reviewing the choices.

For example, for the classes *Student* and *Faculty Member* Adam finds the common hypernym *Person* and creates a new class *Person* as well as subclass relationships between *Student* and *Person*, and between *Faculty Member* and *Person* with Helen's approval. The OWL definition of the hierarchical relationship between *Student* and *Person* is shown

in Figure 5.8.

```
<owl:Class rdf:ID="Person">
<owl:Class rdf:ID="pn.Student">
   <rdfs:subClassOf>
      <owl:Class rdf:ID="#Person"/>
   </rdfs:subClassOf>
</owl:Class>
```

Figure 5.8: Subclass definition in reference ontology

The properties of the base classes are defined next. Property definition starts from the base classes and proceeds to their superclasses. For each base class defined, Adam extracts the property definition entries, such as `map:Student_FirstName a d2rq:PropertyBridge;`, from the Mapping File and defines corresponding base properties of their respective base class. Since the base property names are RDB column names, they may not always be sufficiently descriptive for user level communication. Therefore, Adam presents each base property name to Helen, who can respond in the same three possible ways as described in the case of defining base class names, and the rest of the process is the same. The Mapping File fragment for the properties *FirstName*, *LastName*, and *DOB* is shown in Figure 5.9a; and the OWL definitions of these properties and a meaningful name for *DOB* is shown in Figure 5.9b.

Once the properties of the base classes are defined, Adam and Helen next define the properties of the superclasses. The properties of the superclasses can be defined in two ways. First, Adam conducts a property name comparison to see if there are identical property names among all the subclasses of each superclass, and if there are identical property names He takes the primary names of these properties and defines them as prop-

```
map:Student_FirstName a d2rq:PropertyBridge;
   d2rq:belongsToClassMap map:Student;
   d2rq:property vocab:Student_FirstName;
   d2rq:propertyDefinitionLabel "Student FirstName";
   d2rq:column "Student.FirstName";

map:Student_LastName a d2rq:PropertyBridge;
   d2rq:belongsToClassMap map:Student;
   d2rq:property vocab:Student_LastName;
   d2rq:propertyDefinitionLabel "Student LastName";
   d2rq:column "Student.LastName";

map:Student_DOB a d2rq:PropertyBridge;
   d2rq:belongsToClassMap map:Student;
   d2rq:property vocab:Student_DOB;
   d2rq:propertyDefinitionLabel "Student DOB";
   d2rq:column "Student.DOB";
```

(a) D2RQ Mapping

```
<owl:DatatypeProperty rdf:ID="bn.FirstName">
   <rdfs:domain rdf:resource="#Person" />
   <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="bn.LastName">
   <rdfs:domain rdf:resource="#Person" />
   <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="bn.DOB">
   <rdfs:domain rdf:resource="#Person" />
   <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="pn.Date-Of-Birth">
   </owl:sameAs rdf:resource="#bn.DOB"/>
</owl:DatatypeProperty>
```

(b) OWL

Figure 5.9: Property definition in: (a) Mapping File and (b) reference ontology

erties of the superclass. Second, For each superclass, Adam displays the subclasses along with their properties, asking Helen to specify if there are properties common among the base classes. These common properties may have different primary names even though they refer to the same property of their respective class. Helen may respond in two ways to resolve this name conflict. She may suggest one of the property name to be defined as property of the superclass, or she may suggest a new name representing the common properties. Adam defines the suggested names as properties of the superclass. He then removes the primary names of the common properties from the subclasses because they inherit these properties from their superclass. However, the base property names remain attached to the subclasses. For example, *Person* is a parent class of both *Student* and *Faculty Member*. The properties *FirstName*, *LastName*, and *DOB* are common between the subclasses *Student* and *Faculty Member*; therefore, the primary names of these prop-

erties are moved up the hierarchy and defined as properties of the parent class *Person*.

Finally, the relations between classes are defined. In the Mapping File, a relation between RDB tables is represented by the word *join* followed by the base class names separated by a directed arrow symbol (=>). Adam extracts the base class names appearing in each relation from the Mapping File and creates a graphical picture showing each pair of base classes. Adam displays the graphical picture to Helen, asking her to provide a name for each relation. Adam then defines each relation name as an *ObjectProperty* with the corresponding base class names as the domain and range. (This domain, defined in Section 2.4, differs from domain as an area of knowledge.) For example, Helen provides the word *Offers* to describe the relation between *Department* and *Course*. The Mapping File fragment of this relation is shown in Figure 5.10a, and the OWL definition is shown in Figure 5.10b.

```
map:Course_DepartmentName a d2rq:PropertyBridge;
   d2rq:belongsToClassMap map:Course;
   d2rq:property vocab:Course_DepartmentName;
   d2rq:refersToClassMap map:Department;
   d2rq:join "Course.DepartmentName
   => Department.DepartmentName";
```

(a) D2RQ Mapping

```
<owl:ObjectProperty rdf:about="Offers">
  <rdfs:domain rdf:resource="#pn.Department">
  <rdfs:range rdf:resource="#pn.Course">
</owl:ObjectProperty>
```

(b) OWL

Figure 5.10: Class relation definition in: (a) Mapping File and (b) reference ontology

The synonyms of names are allowed only for the maintenance of backward compatibility between ontology version. For example, a new policy in the university requires *Program* to be called *Department*. As a result, Helen instructs Adam to define a new class name *Department* and designate it as the primary name instead of *Program*. However,

the name *Program* may still be referred to by the users; therefore, Helen asks Adam to define *Program* as a synonym of the new primary name *Department*. The OWL definition of the synonym *Program* is shown in Figure 5.11.

```
<owl:Class rdf:ID="Program">
   <owl:equivalentClass rdf:resource="#pn.Department"/>
</owl:Class>
```

Figure 5.11: Class synonym definition in reference ontology

In the next step, Adam creates a graphical representation of the entire reference ontology and displays to Helen for final approval. A fragment of a university reference ontology illustrating the examples in this scenario is shown in Figure 5.12. Browsing and editing of the reference ontology is facilitated with an ontology editor such as Protégé. By looking at the global picture of the reference ontology Helen may approve or suggest modifications. If Helen approves, Adam completes the construction of the reference ontology and saves it in the Ontology Manager. Otherwise Helen suggests modifications by editing the ontology graph. Adam follows relevant steps to apply the suggested modifications and completes the construction of the reference ontology.

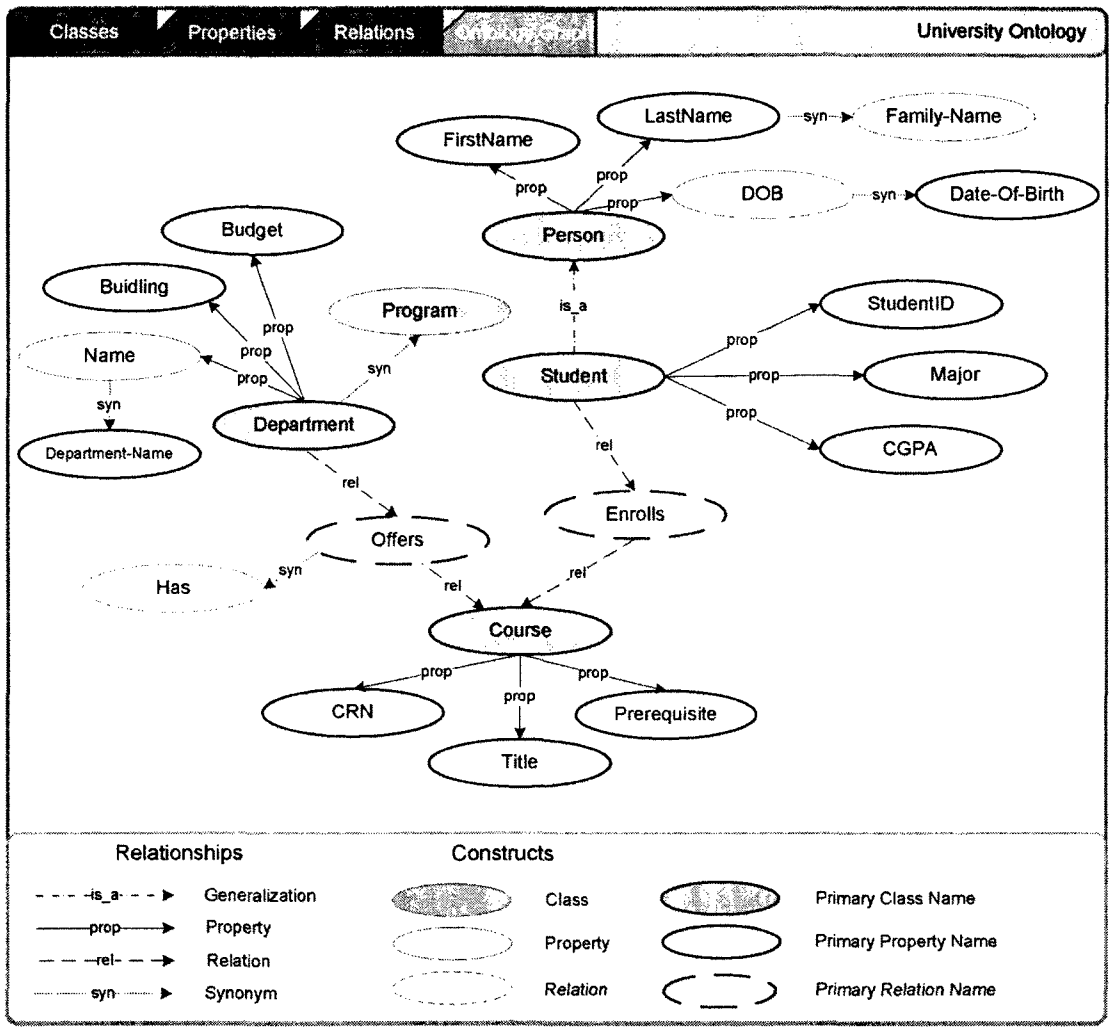

Figure 5.12: Reference ontology graph

5.4 Scenario 3: Developing custom ontology

This scenario shows the processes involved, and the interactions that occur between the user (Henry) and the User Interface Agent (Alice), in developing a custom ontology using the constructs from the reference ontology.

Henry formulates a request for report using the Simplified Natural Language (SNL) and submits it through the User Interface. While processing the request, the SNL Processor performs lexical, syntactic, and semantic analysis, and recognizes the actions that need to be performed and invokes those actions in the appropriate system components. In the case of a query if the request successfully passes through all of the analysis stages, the SNL Processor forwards the statements specifying what information is to be retrieved to the SPARQL Generator, and the statements describing how the retrieved information is to be formatted to the Report Manager.

Assuming that Henry uses a new term called "transfer student" in the request, during semantic analysis the Ontology Manager cannot find the term in the custom or reference ontology; therefore, it posts the following error message in the User Interface Environment (UIE): *New term "transfer student" does not exist.* Upon perceiving this error, Alice engages in a conversation with Henry to clarify the request. Alice displays the following message to Henry: *The term "transfer student" is not found in the ontologies; Please define "transfer student".* In response, Henry enters the following definition: *Transfer student is a student who has transferred credits from previous institution.*

Alice verifies Henry's definition using the ontologies in the next step. Alice extracts the words from the definition and temporarily stores in a word set of *new-term-set.* Using the name of each element in *new-term-set,* Alice searches for matching constructs in the reference ontology. For "transfer student", assuming Alice finds the class construct *Student* and the property construct *Transferred Credits* and displays them to Henry.

If Henry approves the displayed constructs, he also specifies any restriction associated with any of the constructs. In this scenario, Henry approves the class construct *Student*, and the property construct *Transferred Credits* with a restriction that its value cannot be empty. Alice then defines *Transfer Student* as a subclass of the class *Student* and *Transferred Credits* as its property with the restriction that it must have a value (of type number). Note that the class *Student* is in the reference ontology; Alice uses the Unified Resource Identifier (URI) of *Student* as a reference link from the custom ontology. The OWL statements for defining *Transfer Student* in the custom ontology is shown in Figure 5.13. Alice replaces the blank space in *Transfer Student* with a hyphen (-) and appends the leading tag "un" followed by a period (.) to denote that *Transfer Student* is a user-specific name.

If Henry rejects the displayed constructs, Alice asks him to enter a different definition for "transfer student" and follows identical steps in defining the new term.

Once the new term is defined, Ontology Manager creates a graphical representation of the custom ontology and Alice displays it to Henry for final approval. By looking at the global picture of the custom ontology Henry may approve or suggest modifications. If Henry approves, Alice saves it in the Ontology Manger. Otherwise, he suggests modifications by editing the graphical ontology graph. Alice follows relevant steps to apply the suggested modifications in the custom ontology.

```
<owl:Class rdf:ID="un.Transfer-Student">
   <rdfs:subClassOf>
      <owl:Class rdf:ID="#pn.Student"/>
   <rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="pn.Transferred-Credits">
   <rdfs:domain rdf:resource="un.Transfer-Student" />
   <rdfs:range
rdf:resource="&xsd;number"/>
</owl:DatatypeProperty>

<owl:Restriction>
   <owl:onProperty rdf:resource="#pn.Transferred-Credits" />
   <owl:hasValue rdf:datatype="&xsd;number" />
</owl:Restriction>
```

Figure 5.13: Definition of transfer student in custom ontology

In this chapter, I have presented a number of scenarios to demonstrate the behavioral aspects of SRGS. The scenarios have been developed to show how the architecture of SRGS supports the activities involved in accessing information stored in the RDB system and developing ontologies from the RDB schema. In the next chapter, I analyze my work with regards to the main objectives of this thesis.

Chapter 6

Analysis and Evaluation

The main objective of this thesis has been the definition of a system architecture, Semantic Report Generation System (SRGS), that allows developing ontologies from a legacy RDB schema, and accessing information stored in the legacy RDB system. In this chapter, I analyze and evaluate the proposed architecture with respect to these objectives.

The definition of the architectural model presented in chapter 4 begins with a set of system requirements represented as use case diagrams. The system requirements have been carefully developed to guide me in abstracting the aspects of the system that are relevant to my study topic. The basic configuration of the architecture includes a User Subsystem (US) and a Database Subsystem (DBS), each comprised of a software agent and an environment. The software agents assist their human partners in building ontologies and accessing information stored in an RDB system. The environments contain system components designed to provide the functionalities stipulated in the system requirements. The US and the DBS can reside on different machines and communicate through a network.

The basic architecture of SRGS is complete, yet scalable in several aspects. Multiple USs can interact with a single DBS, and multiple DBSs can be attached to a single US. In the US, more instances of the software agent can be created in the event of multiple users accessing the system simultaneously. The presence of multiple agents allows customized assistance for each user's unique requirements in terms of user system interaction and system behavior.

SRGS allows development of ontologies from an RDB structure. In the DBS, the construction of a reference ontology begins with a rudimentary version of reference ontology generated through automatic conversion of the RDB structure to a Semantic Web structure. The converted structure then serves as a starting point from which the Database Interface Agent (DBIA) in interaction with the Database Administrator (DBA) incrementally develops a full reference ontology. In the US, the User Interface Agent (UIA) assists the user in developing a custom ontology, which defines user-specific concepts using constructs from the reference ontology. Thus, SRGS features a novel approach in which software agents assist human partners in developing ontologies.

The agents are equipped with the requisite knowledge of how to build an ontology which includes an understanding of the semantics of general ontological notions, such as class and relationship. In addition, the agents refer to external knowledge resources publicly available on the Semantic Web. The human partners only make decisions and need a good understanding of the ontology building process but owing to the agent assistance need not to have the level of expertise of specialist ontology developers.

The quality of ontologies developed in SRGS depends on two main factors. Though the human partners are not required to become technical experts fully specialized in the ontology development process, their level of understanding of the ontology development process can influence the quality of ontologies to a certain degree. Unintended and accidental errors committed by human partners can be identified and resolved by the agents. However, poor decisions owing to lack of understanding of the ontology development process may result in misrepresentation of knowledge.

The other factor influencing ontology quality is the availability of knowledge resources on the Semantic Web. The agents rely on external knowledge resources, such as lexical dictionaries and libraries of ontologies, on the Semantic Web. The Semantic Web is in its early stage of development, as such these knowledge resources are yet to be realized at large scale. The more such resources are available for agents to exploit the more refined and comprehensive ontologies can be developed.

In Chapter 5, I introduce three scenarios to illustrate the main behavioral aspects of SRGS. The first scenario illustrates how the user of SRGS can access information stored in an RDB using a Simplified Natural Language; the second scenario shows the interactions between the DBIA and the DBA, and the activities that occur within the DBS in the process of developing a reference ontology; and the third scenario demonstrates how the user can complement the reference ontology by introducing user-specific concepts in a custom ontology.

The above analysis suggests that it is possible to combine Semantic Web technologies and MAS approach to create a system architecture for accessing information stored in the RDB system without relying on human intermediaries, and for developing ontologies from an RDB schema. However, these observations remain to be further confirmed through studies involving implementation and experimentation of SRGS.

Chapter 7

Conclusions and Future Work

This thesis proposes and investigates a novel approach to modeling and accessing information stored in legacy RDB systems. The preliminary research has included a review of literature in several areas: legacy RDB systems and their use in decision support; the Semantic Web project and its presently available technology; converting relational data to Semantic Web structures; and multiagent systems (MAS). Those preliminary studies led to several observations. The first observation was that the increasing demands in decision support systems that rely on legacy RDB systems compelled researches to look for more effective techniques that meet modern requirements.

The second observation from the study of the Semantic Web was that information stored in legacy RDB systems can be represented using Semantic Web structures and searched by semantic queries in the SPARQL language; moreover, this can be done on demand, without any modifications of the RDB itself; however high-level semantic interactions between the user and the system require a domain knowledge ontology that is more developed than the rudimentary ontology represented by the RDB schema. The

third observation was that MAS technology holds the promise of development of intelligent decision support systems, with software agents that understand the nature of decision processes; however, this requires that the knowledge underlying the decisions be formally represented as an ontology. The final observation was that the agents themselves can be equipped with a meta-ontology and assist the human partner in the building of domain ontology, which in turn will enable both semantic queries and agent reasoning. These observations have led to the main line of research in this thesis, namely the definition of an architectural model of the Semantic Report Generation System (SRGS) that combines Semantic Web and MAS technologies.

The first step towards defining the architectural model was to formulate a set of system requirements in the form of use cases. These use cases then led to the preliminary definition of the global architecture of SRGS. At the high level, SRGS is comprised of client User Subsystems (US) and server Database Subsystems (DBS). A US consists of a User Interface Environment and a User Interface Agent (UIA). It facilitates user access, processing of users' requests for information, and developing and maintaining custom ontologies. The DBS consists of a Database Interface Environment, a Database Interface Agent (DBIA), and the legacy RDB system. It facilitates developing and maintaining a reference ontology, and retrieving information from the RDB system. The US and DBS can reside on different machines and communicate through a network. Multiple users can simultaneously access the US which can interact with multiple DBSs. Multiple USs can interact with a single or multiple DBSs.

The behavioral aspects of the architecture were then demonstrated through the development of three characteristics scenarios. One scenario shows how the user can directly access information stored in an RDB system through a semantic query, without requiring technical assistance of database programmers and report writers. The other two scenarios illustrate the intelligent assistance of software agents and the specific tasks executed by system components in developing ontologies from the RDB schema.

The analysis of the scenarios suggests that SRGS has met the objective of defining a system architecture that capitalizes on Semantic Web and MAS technologies to create a layer of Semantic Web structures on top of a legacy RDB system in order to facilitate access to information stored in the underlying RDB system. This has been achieved through an innovative combination of Semantic Web and MAS technologies in which agents assist in ontology development.

The next step in the current line of research concerns the possibility of implementing an SRGS prototype that can be used for practical verification of the presented architecture. There are several other issues that can be further researched in order to advance the proposed approach. The software agents can be trained to be able to make more independent decisions and further reduce human involvement in the development of ontologies. The specific steps involved in ontology mediation with regard to importing constructs from external ontologies need to be further studied and elaborated.

Bibliography

S. Ambler. *Agile Database Techniques: Effective Strategies for the Agile Software Developer.* John Wiley & Sons, Inc., New York, NY, USA, 2003.

T. Baker, T. Heath, N. Noy, R. Swick, and I. Herman. Semantic Web Case Studies and Use Cases, 2009. Retrieved June 18, 2011 from `http://www.w3.org/2001/sw/sweo/public/UseCases/`.

F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE.* John Wiley & Sons, Inc., Wiltshire, UK, 2007.

T. Berners-Lee. Semantic Web Road Map. *W3C Design Issues Architectural and Philosophical Points*, 1998a. Retrieved May 03, 2010 from `http://www.w3.org/DesignIssues/Semantic.html`.

T. Berners-Lee. Relational Databases on the Semantic Web. *W3C Design Issues*, 1998b. URL `http://www.w3.org/DesignIssues/RDB-RDF.html`.

T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 285(5):28–37, 2001.

C. Bizer. The Emerging Web of Linked Data. *IEEE Intelligent Systems*, 24:87–92, 2009.

C. Bizer and R. Cyganiak. D2RQ - Lessons Learned. *Position paper for the W3C Workshop on RDF Access to Relational Databases, Cambridge, USA*, 2007.

C. Bizer and A. Seaborne. D2RQ-Treating non-RDF Databases as Virtual RDF Graphs. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, 2004.

C. Bizer, R. Cyganiak, S. Auer, G. Kobilarov, and J. Lehmann. DBpedia - Querying Wikipedia like a Database. In *Developers Track at 16th International World Wide Web Conference (WWW2007), Banff, Canada*, 2007.

C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, September 2009.

R.H. Bordini, M. Wooldridge, and J.F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason.* John Wiley & Sons, 2007.

J.M. Bradshaw, P. Feltovich, and J. Matthew. Human-Agent Interaction. In Guy Boy, editor, *Handbook of Human-Machine Interaction*, pages 283 – 302. Ashgate, 2011.

D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, W3C, 2004. Retrieved October 06, 2010 from `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

D. Brickley and L. Miller. Friend of a Friend Vocabulary Specification, 2005. Retrieved August 28, 2010 from `http://xmlns.com/foaf/spec/`.

J. Broekstra, A. Kampman, and F.V. Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the 1st International Semantic Web Conference*, pages 54–68, Sardinia, Italy, 2002.

K. Byrne. Having Triplets - Holding Cultural Data as RDF. In M Larson, K Fernie, J Oomen, and J Cigarran, editors, *Proceedings of the ECDL 2008 Workshop on Information Access to Cultural Heritage*, volume 1, pages 978–90, Aarhus, Denmark, 2008.

D.D. Chamberlin and R.F. Boyce. SEQUEL: A structured English query language. In *Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control*, pages 249–264, Ann Arbor, Michigan, 1974.

E.F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, 1970.

T. Connolly and C. Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

N. Cullot, R. Ghawi, and K. Yétongnon. DB2OWL : A Tool for Automatic Database-to-Ontology Mapping. In M. Ceci, D. Malerba, L. Tanca, M. Ceci, D. Malerba, and L. Tanca, editors, *Proceedings of the 15th Italian Symposium on Advanced Database Systems (SEBD 2007)*, pages 491–494, Torre Canne, Italy, 2007.

DAML Ontology Library, 2004. Retrieved June 20, 2011 from `http://www.daml.org/ontologies/ontologies.html`.

H.A.A. ElFadeel and A.A.A. ElFadeel. Kngine, 2008. Retrieved January 10, 2011 from `http://www.kngine.com/`.

O. Erling and I. Mikhailov. Mapping Relational Data to RDF in Virtuoso, 2007. Retrieved September 20, 2010 from `http://virtuoso.openlinksw.com/whitepapers/relational%20rdf%20views%20mapping.html`.

D Fensel, F.V. Harmelen, I. Horrocks, D.L. McGuinness, and P.F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2): 38–45, 2001.

M. Graves, A. Constabaris, and D. Brickley. FOAF : Connecting People on the Semantic Web. *Cataloging and Classification Quarterly*, 43(3):191–202, 2007.

A.J.G. Gray, N. Gray, and I. Ounis. Can RDB2RDF Tools Feasibly Expose Large Science Archives for Data Integration? In *Proceedings of the 6th Annual European Semantic Web Conference (ESWC2009)*, pages 491–505, Heraklion, Greece, June 2009.

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

M. Horridge, N. Drummond, S Jupp, G. Moulton, and R. Stevens. A Practical Guide to Building Ontologies Using Protégé 4 and CO-ODE Tools, Editioin 1.2, 2009. The University of Manchester, Manchester, UK.

I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. Semantic Web Architecture: Stack or Two Towers? In *Proceedings of the 3rd International Workshop on Principles and Practice of Semantic Web Reasoning*, pages 37–41, Dagstuhl Castle, Germany, 2005.

IBM. Structured Query Language (SQL), 2006. Retrieved March 22, 2011 from `http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/c0004100.htm`.

N.R. Jennings and M. Wooldridge. Agent-Oriented Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.

G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In *Proceedings of the 11th International Conference on World Wide Web (WWW 02)*, pages 592–603, New York, NY, USA, 2002.

J. Lind. Issues in Agent-Oriented Software Engineering. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, pages 45–58, Limerick, Ireland, 2000.

F. Manola and E. Miller. RDF Primer. Technical report, W3C, 2004. Retrieved May 14, 2010 from `http://www.w3.org/TR/rdf-primer/`.

B. McBride, D. Boothby, and C. Dollin. An Introduction to RDF and the Jena RDF API, 2010. Retrieved June 20, 2011 from `http://openjena.org/tutorial/RDF_API/index.html`.

J. Mccarthy. Ascribing Mental Qualities to Machines. In M. Ringle, editor, *Philosophical Perspectives in Artificial Intelligence*, pages 161–195. Humanities Press, 1979.

D.L. McGuinness and F.V. Harmelen. OWL Web Ontology Language Overview. Technical report, W3C, 2004. Retrieved September 25, 2010 from `http://ia.ucpel.tche.br/~lpalazzo/Aulas/TEWS/arq/OWL-Overview.pdf`.

G.A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38:39–41, 1995.

N. Noy and D.L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical Report KSL-01-05, Knowledge Systems, AI Laboratory, Stanford University, 2001.

C.M. Olszak and E. Ziemba. Approach to Building and Implementing Business Intelligence Systems. *Interdisciplinary Journal of Information, Knowledge, and Management*, 2, 2007.

Ontolingua Server, 2008. Retrieved June 20, 2011 from `http://www.ksl.stanford.edu/software/ontolingua/`.

A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In R. Bordini, M. Dastani, J. Dix, and A.E.F. Seghrouchni, editors, *Multi-Agent Programming*, pages 149–174. Springer Science & Business Media Inc., 2005.

A. Powell, M. Nilsson, A. Naeve, P. Johnston, and T. Baker. Dublin Core Metadata Initiative - Abstract Model, 2007. White Paper Retrieved August 31, 2010 from `http://dublincore.org/documents/abstract-model`.

E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF (Working Draft). Technical report, W3C, 2007. Retrieved August 30, 2010 from `http://www.w3.org/TR/cooluris/`.

F. Ricca, L. Gallucci, R. Schindlauer, T. Dell'Armi, G. Grasso, and N. Leone. OntoDLV: An ASP-based System for Enterprise Ontologies. *Journal of Logic and Computation*, 19:643–670, 2009.

J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual.* Pearson Higher Education, 2nd edition, 2004.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, New Jersey, USA, second edition, 2003.

S.S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau, Jr., S. Auer, J. Sequeda, and A. Ezzat. A Survey of Current Approaches for Mapping of Relational Databases to RDF, 2009. Retrieved July 10, 2010 from `http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf`.

L. Sauermann, R. Cyganiak, D. Ayers, and M. Völkel. Cool URIs for the Semantic Web, 2008. Retrieved April 29, 2010 from `http://www.dfki.uni-kl.de/~{}sauermann/2006/11/cooluris/`.

A. Seaborne. RDQL - A Query Language for RDF (Member Submission). Technical report, W3C, 2004. Retrieved August 30, 2010 from `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/`.

J.F. Sequeda, S. Tirmizi, and D. Miranker. A Bootstrapping Architecture for Integration of Relational Databases to the Semantic Web. In *Proceedings of the 7th International Semantic Web Conference (ISWC2008)*, Karlsruhe, Germany, October 2008.

J.F. Sequeda, R. Depena, and D. Miranker. Ultrawrap: Using SQL Views for RDB2RDF. In *Proceedings of the 8th International Semantic Web Conference (ISWC2009)*, Washington, DC, USA, 2009.

Y. Shoham. Agent Oriented Programming. *Artificial Intelligence*, 60:51–92, March 1993.

M. K. Smith, C. Welty, and D.L. Mcguinness. Owl web ontology language guide, 2004. Retrieved June 27, 2011 from `http://www.w3.org/TR/owl-guide/`.

I. Sommerville. *Software Engineering.* Pearson/Addison Wesley, USA, 2004.

D. Steer. SquirrelRDF, 2009. Retrieved September 05, 2010 from `http://jena.sourceforge.net/SquirrelRDF/`.

V. Tamma and T.R. Payne. Is a Semantic Web Agent a Knowledge-Savvy Agent? *IEEE Intelligent Systems*, 23:82–85, 2008.

J. Tao, E. Sirin, J. Bao, and D.L McGuinness. Integrity constraints in owl. In *Proceedings of the 24th Conference on Artificial Intelligence(AAAI 2010)*, Atlanta, GA, USA, 2010.

TopQuadrant Inc. TopBraid Composer 2007: Getting Started Guide Version 2.0, 2007. Retrieved June 16, 2011 from `http://www.topquadrant.com/docs/marcom/TBC-Getting-Started-Guide.pdf`.

M. Wooldridge. *An Introduction to Multiagent Systems.* Wiley, Glasgow, UK, 2nd edition, 2009.

M. Wooldridge and N.R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

Appendix A

The D2RQ Platform

The D2RQ Mapping File is explained in A.1, and the proposed extension to the D2RQ Platform is described in A.2.

A.1 The D2RQ Mapping File

The Mapping File contains the RDF representation of an RDB schema. Its file format is W3C standard format Notation 3 (N3), which is a compact alternative to the RDF syntax, intended for human readability and designed to optimize expression of data and logic in the same language. The Mapping File can be generated by running the `generate-mapping` script available in the D2RQ Platform software package. When this script is run, the D2RQ Engine analyzes the schema of the database and creates an RDF representation of the schema. The D2RQ Platform then uses the Mapping File every time it translates RDB data to RDF triples. An excerpt of the Mapping File generated from an RDB called university is given below.

```
@prefix map: <file:/home/rdb2rdf/d2r-server-0.7/university.n3#> .
@prefix vocab: <http://localhost:2020/vocab/resource/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .

map:database a d2rq:Database;
d2rq:jdbcDriver "com.mysql.jdbc.Driver";
d2rq:jdbcDSN "jdbc:mysql://localhost/university";
d2rq:username "root";
d2rq:password "123456";
jdbc:autoReconnect "true";
jdbc:zeroDateTimeBehavior "convertToNull";
.

# Table Course
map:Course a d2rq:ClassMap;
d2rq:dataStorage map:database;
d2rq:uriPattern "Course/@@Course.CRN|urlify@@";
d2rq:class vocab:Course;
d2rq:classDefinitionLabel "Course";
.
map:Course_CRN a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:Course;
d2rq:property vocab:Course_CRN;
d2rq:propertyDefinitionLabel "Course CRN";
d2rq:column "Course.CRN";
.
map:Course_Title a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:Course;
d2rq:property vocab:Course_Title;
d2rq:propertyDefinitionLabel "Course Title";
d2rq:column "Course.Title";
.
map:Course_DepartmentName a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:Course;
d2rq:property vocab:Course_DepartmentName;
d2rq:refersToClassMap map:Department;
```

```
d2rq:join "Course.DepartmentName => Department.DepartmentName";
.

# Table Department
map:Department a d2rq:ClassMap;
d2rq:dataStorage map:database;
d2rq:uriPattern "Department/@@Department.DepartmentName|urlify@@";
d2rq:class vocab:Department;
d2rq:classDefinitionLabel "Department";
.
map:Department_DepartmentName a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:Department;
d2rq:property vocab:Department_DepartmentName;
d2rq:propertyDefinitionLabel "Department DepartmentName";
d2rq:column "Department.DepartmentName";
.
map:Department_Building a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:Department;
d2rq:property vocab:Department_Building;
d2rq:propertyDefinitionLabel "Department Building";
d2rq:column "Department.Building";
.
```

The Mapping File begins with the declaration of a number of prefixes for the common Unified Resource Identifiers (URI). Of particular interest is the base URI: `<http://localhost:2020/vocab/resource/>`, which establishes a vocabulary namespace for the constructs in the Mapping File. When each vocabulary is given a namespace, the ambiguity between identically named elements across multiple vocabularies can be resolved.

The `d2rq:Database` tag specifies the JDBC connection to the database and the login credentials for accessing the database. The `d2rq:ClassMap` tag represents an RDB table as class, and `d2rq:PropertyBridge` represents an RDB column as property. A relation-

ship between two RDB tables is specified by the `d2rq:join` tag. For example, the tag `d2rq:join "Course.Department_Name => Department.Department_Name"` defines the relation between the Course and Department tables, i.e., department offers course.

A.2 D2RQ extension

I installed the D2RQ Platform and tested it against a small database named *university*. I ran **find(s p o)** SPARQL query, and the D2R Server displayed correct results. While the server was running, I changed a data value in a table and D2R server showed the new value in real time. This proved that the D2RQ Platform provides on-demand Relational Database (RDB) to Resource Description Framework (RDF) mapping. However, my test revealed a flaw in the platform. It failed to detect any changes that were made to the database schema. For instance, when I added a new column to the a table and populated it with new data, find(s p o) query results did not include the newly added column and its values. When I dropped or renamed a column the D2R Server gave the following error:

> Unknown column 'Department.Budget' in 'field list': SELECT DISTINCT 'Department'.'Budget', 'Department'.'Department_Name' FROM 'Department' (E0)

In this case, I dropped the Budget column from the Department table. Creating or dropping a table resulted in similar errors. Moreover, I encountered the error message shown in Figure A.1 when I stopped the D2R Server and tried to launch again.

```
File Edit View Terminal Help
rg.mortbay.log.Slf4jLog
20:02:15 INFO  log                  :: jetty-6.1.10
20:02:15 INFO  log                  :: NO JSP Support for , did not find org.apache.jasper.servlet.JspServ
let
20:02:16 INFO  D2RServer            :: using port 8080
20:02:16 INFO  D2RServer            :: using config file: file:/home/mohammad/rdb2rdf/d2r-server-0.7/cours
e schedule_mapping.n3
20:02:16 ERROR log                  :: Failed startup of context org.mortbay.jetty.webapp.WebAppContext@68
2406{,webapp}
de.fuberlin.wiwiss.d2rq.D2RQException: Column @@Department.Budget@@ not found in database (E0)
        at de.fuberlin.wiwiss.d2rq.dbschema.DatabaseSchemaInspector.columnType(DatabaseSchemaInspector.jav
a:96)
        at de.fuberlin.wiwiss.d2rq.sql.ConnectedDB.columnType(ConnectedDB.java:317)
        at de.fuberlin.wiwiss.d2rq.map.Mapping$AttributeTypeValidator.validate(Mapping.java:173)
        at de.fuberlin.wiwiss.d2rq.map.Mapping.validate(Mapping.java:96)
        at de.fuberlin.wiwiss.d2rq.GraphD2RQ.<init>(GraphD2RQ.java:85)
        at de.fuberlin.wiwiss.d2rq.GraphD2RQ.<init>(GraphD2RQ.java:74)
        at de.fuberlin.wiwiss.d2rq.ModelD2RQ.<init>(ModelD2RQ.java:61)
        at de.fuberlin.wiwiss.d2rs.AutoReloadableDataset.initD2RQDatasetGraph(AutoReloadableDataset.java:8
0)
        at de.fuberlin.wiwiss.d2rs.AutoReloadableDataset.forceReload(AutoReloadableDataset.java:54)
        at de.fuberlin.wiwiss.d2rs.D2RServer.start(D2RServer.java:225)
        at de.fuberlin.wiwiss.d2rs.WebappInitListener.contextInitialized(WebappInitListener.java:37)
        at org.mortbay.jetty.handler.ContextHandler.startContext(ContextHandler.java:540)
        at org.mortbay.jetty.servlet.Context.startContext(Context.java:135)
        at org.mortbay.jetty.webapp.WebAppContext.startContext(WebAppContext.java:1220)
        at org.mortbay.jetty.handler.ContextHandler.doStart(ContextHandler.java:510)
        at org.mortbay.jetty.webapp.WebAppContext.doStart(WebAppContext.java:448)
        at org.mortbay.component.AbstractLifeCycle.start(AbstractLifeCycle.java:39)
        at org.mortbay.jetty.handler.HandlerWrapper.doStart(HandlerWrapper.java:130)
        at org.mortbay.jetty.Server.doStart(Server.java:222)
        at org.mortbay.component.AbstractLifeCycle.start(AbstractLifeCycle.java:39)
        at de.fuberlin.wiwiss.d2rs.JettyLauncher.start(JettyLauncher.java:64)
        at d2r.server.startServer(server.java:86)
        at d2r.server.main(server.java:57)
20:02:16 INFO  log                  :: Started SocketConnector@0.0.0.0:8080
Exception in thread "main" java.lang.NullPointerException
        at de.fuberlin.wiwiss.d2rs.JettyLauncher.start(JettyLauncher.java:68)
        at d2r.server.startServer(server.java:86)
        at d2r.server.main(server.java:57)
```

Figure A.1: D2R Server error

After I had replaced the old Mapping File with a new version, I was able to launch the D2R Server. I also noticed my schema changes appeared in find(s p o) query results. Therefore, I came to the conclusion that in order to achieve real time consistency between RDB data and SPARQL query results, the Mapping File needs to be recreated whenever the RDB schema is modified. In order to automate this process I have added an extension to the D2RQ Platform. The conceptual picture of the proposed extension

is shown in Figure A.2; and the processes in mapping RDB data to RDF triples with the extension in place are illustrated in Figure A.3.

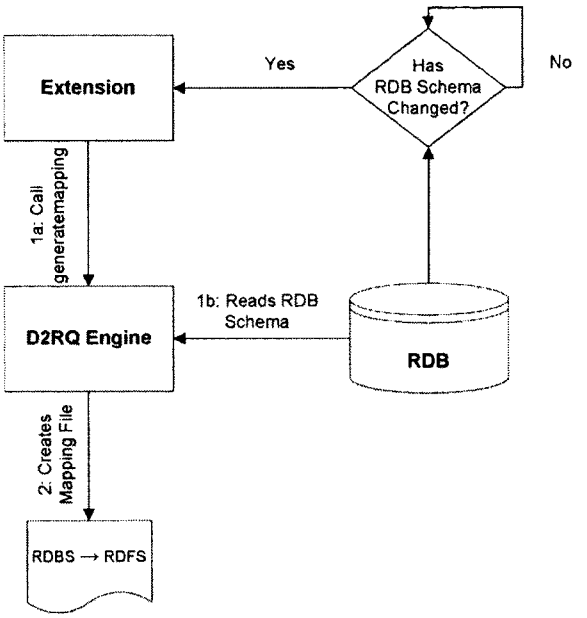

Figure A.2: The extension

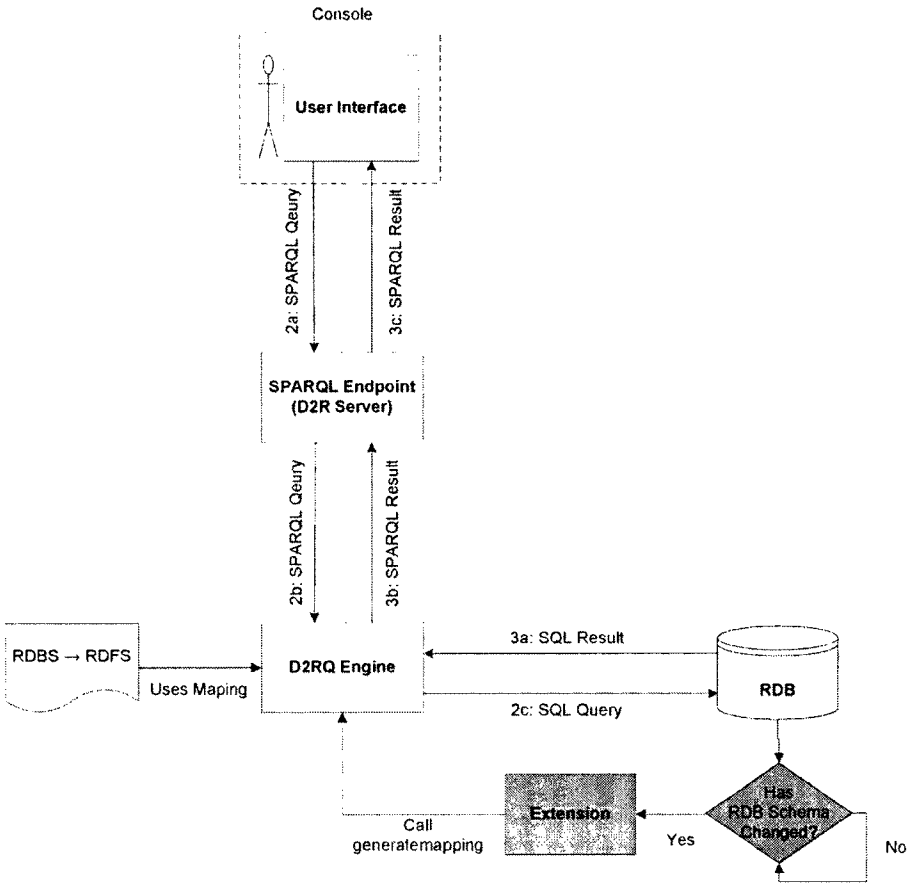

Figure A.3: The D2RQ Platform with the extension

The extension can be implemented using one of the following two methods

Binary log processing

MySQL generates binary log files that record every transaction occurring in the databases. A log file can be associated to a database, and MySQL updates the log file every time a query is executed on that database. My proposed extension analyzes the log file contents to find out whether the most recent transaction has modified the

database schema. I take the most recently executed query from the log file and run it through a string tokenizer to search for CREATE, ALTER or DROP string, because a query with one of these SQL statements is the one that modifies the database schema. When a match is found, the D2RQ Platform is invoked to update the Mapping File. This process is illustrated in figure A.4.

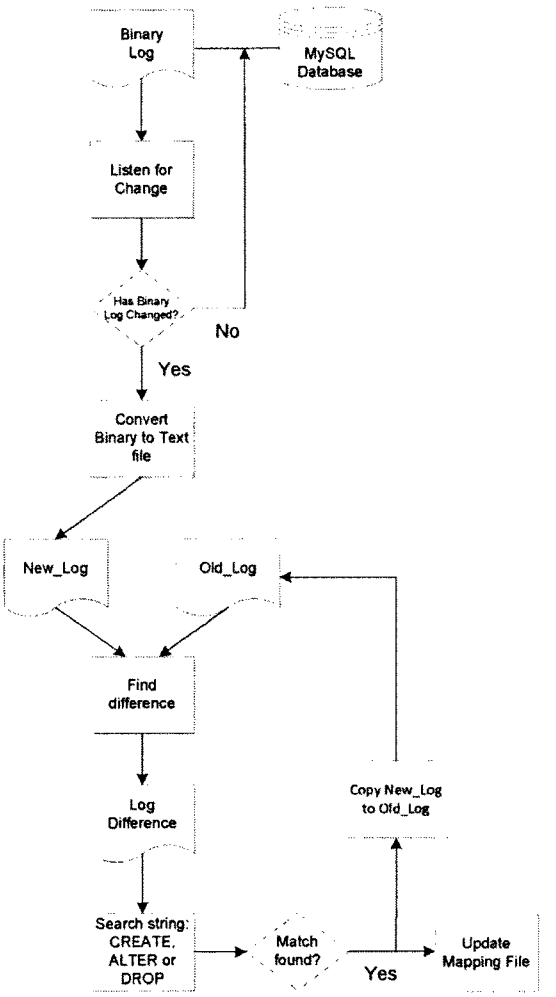

Figure A.4: Binary log processing method

Query interception

MySQL Proxy is a free and open source application that can intercept all queries and responses between a MySQL client and server. The extension uses MySQL Proxy to intercept all incoming queries, it then checks whether a query has the string CREATE, ALTER or DELETE in it. If the extension finds a match it invokes the D2RQ Platform to update the Mapping File. This method is illustrated in Figure A.5.

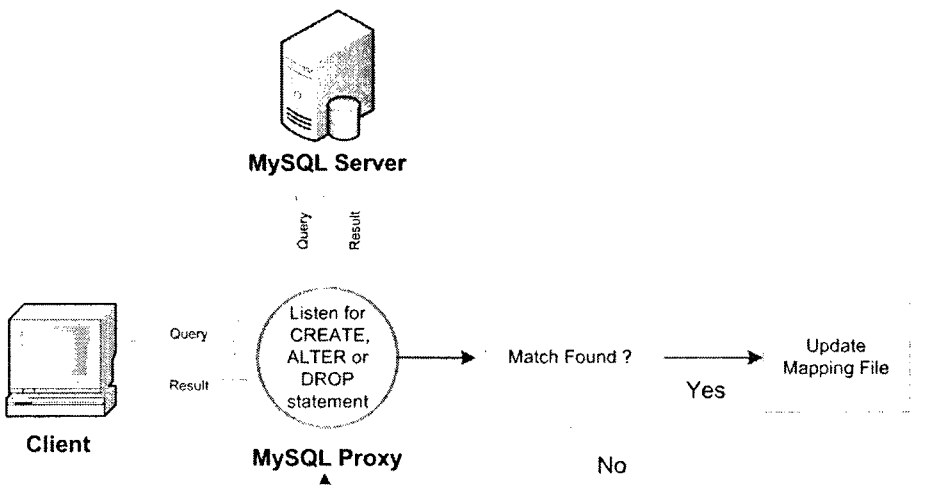

Figure A.5: Query interception method