A Multiagent Architecture for Semantic Access to Legacy Relational Databases

Desanka Polajnar
Department of Computer Science
University of Northern
British Columbia
Prince George, BC
Canada, V2N 4Z9
Email: desa@unbc.ca

Mohammad Zubayer
Department of Computer Science
University of Northern
British Columbia
Prince George, BC
Canada, V2N 4Z9
Email: zubayer@unbc.ca

Jernej Polajnar
Department of Computer Science
University of Northern
British Columbia
Prince George, BC
Canada, V2N 4Z9
Email: polajnar@unbc.ca

Abstract—**This paper presents a novel approach to accessing information stored in legacy relational databases (RDB), based on Semantic Web (SW) and multiagent systems (MAS) technologies. Its purpose is to provide the users of enterprise decision-support systems with direct, flexible, and customized access to information, through high-level semantic queries, without the need to modify the underlying legacy databases. This is effected through a distributed architecture, based on agent-oriented intelligent middleware, consisting of servers that contain databases and clients that provide customized access to users. In a server, the meaning of the database structure is captured in the reference ontology, which is exported to clients. In a client, the meaning of concepts specific to a user is captured in the custom ontology of that user, as an overlay that relies on the imported reference ontologies. These system ontologies are built gradually, within the system itself, in a process that automatically generates the basic structures from RDB schemas and incrementally enhances them to full ontologies through human-agent interactions. The approach relies on agents endowed with meta-ontology that assist human actors in the ontology-building process. A part of their role is to find, identify, reference, import, display, and apply relevant knowledge available on the Semantic Web. The development of ontologies in turn permits further delegation of operational tasks to agents. The approach is expected to become increasingly effective with the advancement of the SW infrastructure. The paper explains the conceptual basis of the approach, outlines the distributed system architecture, and focusses on the server functionality, including the development of reference ontology.**

I. Introduction

The impact of computer-based information systems on the progress of human society is well acknowledged in all disciplines. 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 compel researchers to look beyond traditional access techniques in order to meet modern requirements.

In legacy information systems, the relational database model 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, which involves 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 [1]. That approach still requires human mediation, time to restructure large amounts of data, and accurate foresight as to what information might be needed.

In this paper, we explore an alternative approach, aimed at overcoming those limitations. It combines two fast-developing technologies – the Semantic Web [2] and multiagent systems (MAS) [3] – to provide the users of enterprise decision-support systems with direct, flexible, and customized access to information, through high-level semantic queries. Since it does not require modification of underlying databases, our proposed form of access can co-exist with the more traditional ones. It allows continuous use of the legacy system. We present the approach in the form of an intelligent distributed system, called the Semantic Query Access System (SQAS), with servers containing databases and clients providing access to users. Its basic functionality is provided by agent-oriented middleware.

Many of the issues arising in SQAS are closely related to Semantic Web research. The SW project envisions a world-wide infrastructure providing universal integrated semantic access to a multitude of distributed knowledge sources. This requires a hierarchy of standard ontologies that correspond to various knowledge domains at different levels of abstraction, as well as languages, design techniques, software components, and tools. As SW technology matures, many of the SQAS development needs should be satisfiable from its repository. Differrences stemming from the "closed world" [4] nature of enterprise systems (vs. "open world" SW) are also being studied (e.g., [5]). However, ontologies representing the meaning of database structures in SQAS must be specifically developed. In our approach that is done within the system itself.

An innovative feature of SQAS is the role of agents in ontology building. The system ontologies are built gradually. In a server, the meaning of the database structure is captured in the *reference ontology*. This includes automatic generation of the basic structures from RDB schemas and their incremental enhancement to full ontology through human-agent interactions. Reference ontologies are exported to clients that need them. In a client, a layer of *custom ontology* is constructed for each user, as an overlay that relies on the imported reference ontologies, again through human-agent interactions. The approach relies on agents endowed with meta-ontology that assist human actors in the ontology-building process. A part of their role is to find, identify, reference, import, display, and apply relevant knowledge available on the Semantic Web. The development of ontologies in turn permits further delegation of operational tasks to agents. The approach is expected to become increasingly effective with the advancement of the SW infrastructure.

The rest of the paper describes the system objectives (Section II), the basic distributed architecture (III), the components constituting the intelligent middleware, their roles and interactions (IV), the process of agent-assisted development of reference ontology (V), and the conclusions (VI).

II. System objectives

In formulating the system objectives, we are primarily interested in the viewpoint of the decision-making user who accesses information in an existing relational database (RDB) system. The user is aware that the structure of the database and its contents may evolve over time. Within the framework of our current model, the user does not cause or control such changes. The system is situated in an institutional or corporate environment. The user is assumed to be familiar with the knowledge domain of the information in the database, but may differ in specific interests from other users of the same system.

We start by describing the system requirements in terms of use cases and actors. A use case is 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 [6].

The system requirements are developed in three steps. We first describe 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. We then focus on on user-system interaction in legacy RDB systems. Finally, we describe the requirements for user-system interaction in the Semantic Query Access System (SQAS), introduced in this paper. In that context we then elaborate the objectives of SQAS and its perceived practical advantages.

A. The generic system

The actors and high-level use cases of the generic system are shown in Fig. 1. The actors of primary interest for us are the User and the Database Administrator (DBA). The use cases are largely self-explanatory. The top four use cases of Fig. 1 capture the generic system functions performed on behalf of the user, regardless of how these functions are implemented. The last two enable the DBA and the Data Entry Operator (DEO) to maintain the RDB structure and content respectively. We next focus on two use cases, *Accept Request for Information and Present Report* and *Manage Ontology*, in order to highlight the differences between legacy systems and SQAS.

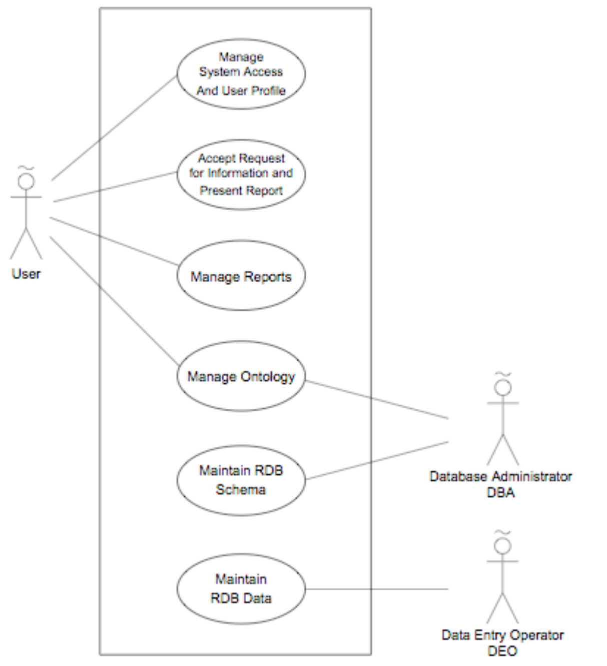

Manage System Access And User Profile
Accept Request for Information and Present Report
Manage Reports
Manage Ontology
Maintain RDB Schema
Maintain RDB Data
User
Database Administrator DBA
Data Entry Operator DEO

Fig. 1. The actors and high-level use cases of the generic system

B. Legacy RDB system

In a legacy RDB system, some of the functionalities of the generic system shown in Fig. 1 are performed by a human intermediary, the report writer, on behalf of the decision-making user, while others are performed by the computer system. The actors and the basic high-level use cases of a legacy RDB system are shown in Fig. 2. Let us elaborate the two key generic use cases.

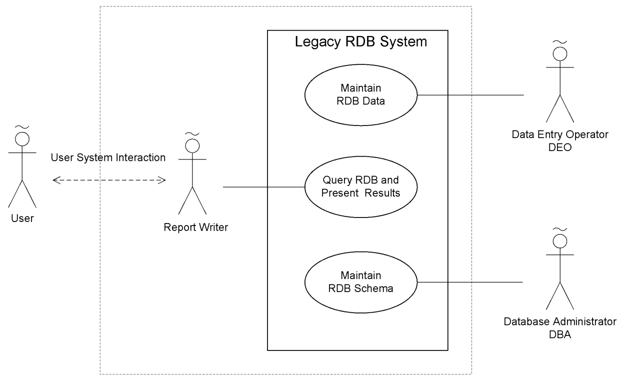


Fig. 2. Legacy RDB system

1) Accept Request for Information and Present Report: The report writer accepts a request, in which the user specifies what information should be retrieved and how it should be presented; the report writer then queries the RDB to retrieve the information and presents it to the user in the requested format. If the meaning of the request is not clear, the report writer interacts with the user in natural language to clarify it.

2) Manage Ontology: This use case is concerned with the correspondence between the database structures and their meaning. Apart from the basic relationships captured within the RDB schema, the meaning is typically captured informally in natural-language documentation (and sometimes human knowledge) maintained by the DBA. The User's terminology and conceptual framework may differ from the ones presented by the DBA. In order to learn both, the report writer typically depends on informal documentation, and on natural-language interactions with the User and the DBA.

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

C. The Semantic Query Access System (SQAS)

In SQAS, the user directly interacts with the system that performs the functionalities in the top four use cases of Fig. 1, eliminating the report writer role. We briefly describe the key generic use cases.

1) Accept Request for Information and Present Report: This use case allows the user to directly communicate report requests to the system, in simplified natural language. 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, similar to the report writer in a legacy RDB system. It then retrieves the information and presents in the requested format.

2) Manage Ontology: The meaning of database structures is formally captured in the *reference ontology*. The reference ontology represents knowledge originating from the underlying relational database structure, the human actors in the system, and external ontologies 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 terminology and conceptual framework of the User are formally captured in the *custom ontology*. The custom ontology is a layer on top of the reference ontology that is custom-built for each specific user. It is also built within the system itself, in interaction with the User, with access to the reference ontology and external ontologies available on the Semantic Web. The User actor now has the additional role of managing the custom ontology.

D. Decomposition of SQAS use cases

The functions of each high-level use case can be further specified through decomposition into simpler use cases. In presenting the decompositions of key generic use cases in SQAS, we also decompose the functionality into its client part, related to the User, and its server part, related to the RDB.

1) Accept Request for Information and Present Report: The decomposition of this use case is shown in Fig. 3. In general, a client can interact with multiple servers, and a server with multiple clients. In the client, the *Accept Request for Information (SNL) and Present Results* use case allows the user to formulate a 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 checks that each statement as a whole in the request is semantically correct. If the SNL request is valid, the *Generate SPARQL Script* use case creates a SPARQL script from the intermediate representation of the request. The client then sends the SPARQL script to the server, and receives the SPARQL results from it. The SPARQL results are then formatted and presented by the *Format and Display Report* use case.

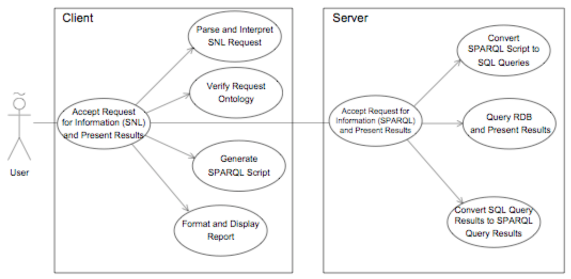

Client
Parse and Interpret SNL Request
Verify Request Ontology
Accept Request for Information (SNL) and Present Results
Generate SPARQL Script
Format and Display Report
User
Server
Convert SPARQL Script to SQL Queries
Accept Request for Information (SPARQL) and Present Results
Query RDB and Present Results
Convert SQL Query Results to SPARQL Query Results

Fig. 3. Use case: *Accept request for information and present report*

In the server, the *Accept Request for Information (SPARQL) and Present Results* use case receives the SPARQL script and has it translated to equivalent SQL queries by 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 for translation. The *Accept Request for Information (SPARQL) and Present Results* use case sends the results to the client.

2) Manage Ontology: The decomposition of the *Manage Ontology* use case is shown in Fig. 4. The use cases are grouped by their functionalities into use cases that directly communicate with the User, the client; and use cases that communicate with the DBA, the server.

The client invokes the *Import Reference Ontology* use case when it connects to the server, relying on the functions of the *Export Reference Ontology* use case in the server. The *Initialize Custom Ontology* use case allows the user to create a conceptual framework specific to the user. The *Update Custom Ontology* use case lets the user modify definitions of user-specific concepts in the custom ontology. When the reference ontology is updated in the server, the *Maintain Consistency of Reference Ontology* use case ensures that the updates are also applied to the reference ontology in the client. The reference

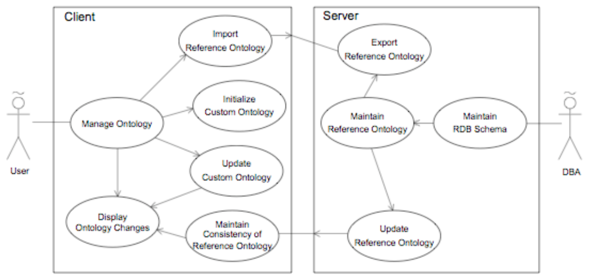


Fig. 4. Use case: *Manage ontology*

ontology updates are displayed to the user by the *Display Ontology Changes* use case.

In the server, the *Export Reference Ontology* use case sends a copy of the reference ontology to the attached client. The *Maintain RDB Schema* use case allows the DBA to modify the structure of the RDB. 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.

III. Architecture

A. The general architecture of SQAS

At the high level, SQAS consists of two types of subsystems: *User Subsystem (US)*, representing the client functionality, and *Database Subsystem (DBS)*, representing the server functionality. These subsystems normally reside on different machines, connected through a wide area network, using a standard transport protocol. In general, a US can connect to multiple DBSs, and a DBS can serve multiple USs. In addition, a single US can host multiple human users.

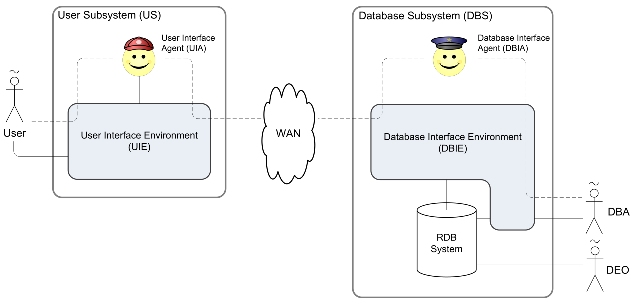


Fig. 5. The basic architecture of SQAS

The basic architecture consists of a single US and a single DBS, with a single user accessing the system. This configuration is depicted in Fig. 5 and used in the sequel to discuss the design of SQAS. In this paper we note but do not discuss in detail the ontology management issues arising from the presence of multiple users, multiple USs, and multiple DBSs.

B. Agent roles

This subsection describes the specific agent roles in SQAS.

1) User Interface Agent:

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. If the SNL processor in any stage of statement analysis produces a warning, the UIA tries to autonomously resolve arising issues in interaction with the subsystem components. If the SNL processor produces an error, the UIA engages with the user to correct the error in the statements.

Searching the Semantic Web. The agent can search the Semantic Web for relevant external ontologies. For instance, it can look up synonyms and hypernyms of terms in a language ontologies such as WordNet [7]. It can also look for relevant domain ontologies to standardize the usage of terms or complement the locally developed custom ontology.

Development of custom ontology. The UIA helps the user create and maintain a custom ontology, which describes the conceptual framework specific to the user and is directly translated to the reference ontology. In order to keep the two ontologies consistent, the UIA ensures that any update in the reference ontology is also reflected in the custom ontology.The UIA has the technical knowledge of the ontology development process, including the use of external ontologies.

Customizing the behavior of User Interface. While assisting the user with SNL dialogue, the UIA can observe if the user is typically making certain types of choices, and offer this choice first in their 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.

Coordination of reference ontologies. There may be multiple reference ontologies if the US is attached to multiple DBSs. The UIA needs to resolve any conflicts between imported reference ontologies.

When multiple users operate within the same US, there is a different UIA for each user.

2) Database Interface Agent:

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 agent's actions are similiar as in US.

Searching the Semantic Web. The agent's actions are similar as in US, but with primary emphasis on ontological sources needed in the building of reference ontology.

Development of reference ontology. The DBIA interacts with the DBA in developing and maintaining a reference ontology. The *Schema to Base Ontology Mapper* analyzes the RDB schema and generates a Mapping File, which contains RDF models of the RDB schema. The Mapping File then serves as the *base ontology* from which the DBA incrementally builds a full reference ontology with the assistance of the agent. The DBIA provides technical guidance in the ontology development process, and access to ontological resources on the Semantic Web.

Customizing the behavior of User Interface. As in US.

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. Depending on the performance requirements, DBIA can be replaced with a team of agents.

IV. Agent-oriented middleware

This section describes the structure of software within the US and DBS. At each end, it consists of an agent and its environment that contains a set of interacting software components. The environment components can be designed and implemented using the conventional object-oriented software engineering (OOSE) methodology, with an emphasis on efficient performance. The agent can observe every component and interact with it, and it also interacts with the primary human actor. While these interactions are central to our discussion in the present paper, it is important to note that the agents along with core environment components constitute a layer of *intelligent middleware* that can offer support to other applications. An enterprise system normally includes a variety of business intelligence applications performing various types of analysis. In the context of SQAS, such applications would be realized as agent-oriented software, running on top of core SQAS. In general, agents in such applications would interact with the SQAS agents described in this paper, with the user, and with the Semantic Web.

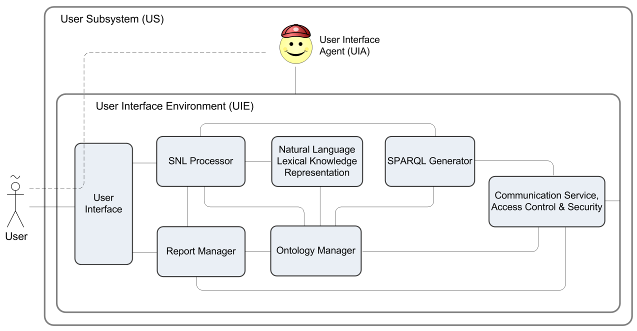
A. The User Subsystem

The architectural structure of the US is shown in Fig. 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 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 system objectives without additional human assistance.

1) The User Interface Environment (UIE): All the components that communicate with the user and the UIA are grouped into the UIE. The solid lines represent direct communication between the user, the components, and the UIA. The dashed line represents communication between the user and the UIA. The UIE consists of the following main components:

The *User Interface (UI)* enables all communications between the user and the system. It provides the functionalities with regard to accessing the system as formulated in II-C.

The *SNL Processor* component enables the user to interact with the system using SNL. The user formulates requests for information and modifications to the custom ontology using SNL statements. The SNL processor first performs the lexical analysis and syntax analysis, using the language definition and vocabulary information from the custom and reference ontologies. After that, it performs semantic analysis, again checking that the relationships between concepts in the statement are consistent with the custom and reference ontologies, to ensure that each statement as a whole is meaningful. Once the intermediate representation is successfully generated, the SNL Processor invokes the relevant components. 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.


User Subsystem (US)
User Interface Agent (UIA)
User Interface Environment (UIE)
User Interface
SNL Processor
Natural Language Lexical Knowledge Representation
SPARQL Generator
Communication Service, Access Control & Security
Report Manager
Ontology Manager
User
Fig. 6. The User Subsystem

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.

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.The Report Manager allows the user to view, reformat, save, and delete reports.

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. Updates to the reference ontology may require updates to the custom ontology in order to maintain consistency. The updating process may require the involvement of UIA, and possibly the User.

The *Natural Language Lexical Knowledge Representation* 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 acess to external language ontologies. The UIA and the SNL Processor communicate with this component to look up meanings and relationships between natural language terms.

The *Communication Service, Access Control and Security* component facilitates communication between the US and the DBS. It provides sser authentication, privileges, security, and the interactions with lower-layer communication services.

2) The User Interface Agent (UIA): The UIA can interact with the User and with all components within the UIE, in performing its role described in III-B.

B. The Database Subsystem

The architectural structure of the DBS is shown in Fig. 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.

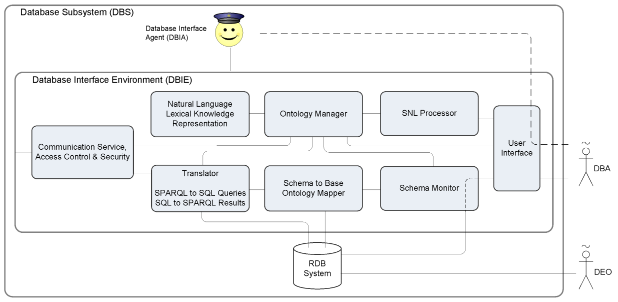

Database Subsystem (DBS)
Database Interface Agent (DBIA)
Database Interface Environment (DBIE)
Natural Language Lexical Knowledge Representation
Ontology Manager
SNL Processor
Communication Service, Access Control & Security
User Interface
Translator
SPARQL to SQL Queries
SQL to SPARQL Results
Schema to Base Ontology Mapper
Schema Monitor
DBA
RDB System
DEO

Fig. 7. The Database Subsystem

1) The Database Interface Environment (DBIE): All the components that communicate with the DBIA, the DBA, and the DEO are grouped into the DBIE. The solid lines represent direct communication between the components. The dashed line represents communication between the DBA and the DBIA. The DBIE consists of the following main components:

The *User Interface* 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.

The *SNL Processor* 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 role of the SNL processor is similar as in US.

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.

The *Natural Language Lexical Knowledge Representation* 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.

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.

The *Schema to Base Ontology Mapper* 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.

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 to reflect the modifications in the reference ontology.

The *Communication Service, Access Control and Security* 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.

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

2) Database Interface Agent: The Database Interface Agent (DBIA) can interact with the DBA and with all components within the DBIE, in performing its role described in III-B.

V. Agent-assisted ontology development

In this section, we analyze and illustrate the behavior of the SQAS agents as ontology builders. The notion of ontology development by software agents is not mentioned in the literature we have reviewed; it is a novel aspect of this approach. Scenarios involving both custom and reference ontologies have been elaborated in [8]. Below we present a few illustrative fragments.

In SQAS, 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 SQAS agents must have the requisite knowledge of how to build an ontology in order to fulfill their roles 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*. 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 knowledge domains specific to the given databases.

An ontology language is required to formally define classes, properties and relationships. This paper uses Web Ontology Language (OWL) for illustration, as it is the commonly known standard ontology language for the Semantic Web, even though some of its properties, such as the Open World Assumption (OWA), make it less suitable for modeling ontology developed from an RDB schema than, for instance, OntoDLV [5]. The questions related to the optimal choice of ontology language for SQAS are beyond the scope of this paper.

In SQAS, there can be multiple names describing the same instance of a given concept. Among them, a *primary name* is designated by the DBA as the official name of the concept instance in the reference ontology; each concept instance derived from the RDB schema also inherits a *base name* from the RDB schema concept, 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 primary name. The system tolerates the usage of other possible names by human actors, but agents use the primary name in reasoning and interactions.

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

Fig. 8. A *class* definition in Mapping File

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

Fig. 9. A *class* definition in reference ontology

In SQAS, agents reason with their own meta-ontologies, ontologies developed within SQAS, and public ontologies on the Semantic Web. The agents adopt the Closed World Assumption (CWA) when reasoning with ontologies that are within the boundary of SQAS, 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 [5].

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

Fig. 10. A *subclass* definition in reference ontology

The SQAS agents are also equipped with the know-how of specific steps involved in the ontology construction [9]. In the scenario that follows, the methodological steps in developing ontology have been adapted for SQAS. In SQAS, an agent in

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

Fig. 11. A *property* definition in Mapping File

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

Fig. 12. A *property* definition in reference ontology

interaction with a human partner builds a reference ontology from an RDB schema. In this process, the D2RQ Platform [10] 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. For illustration, we use s small example involving a student database.

```
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";
```

Fig. 13. A *class relation* definition in Mapping File

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

Fig. 14. A *class relation* definition in reference ontology

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 (Fig. 8,11), and uses them to define the corresponding base classes, properties in the reference ontology (Fig. 9,12) . It then interacts with the human partner to extend the ontology with more general classes (Fig. 10), and to introduce the properties with meaningful names. The agent also extracts from the Mapping File the relations between classes (Fig. 13), and with the help of the human partner defines the corresponding class-relations in the reference ontology (Fig. 14). 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. When the building of reference ontology is completed, the DBIA dispays it as an editable graph to the DBA for modification and approval (Fig. 15).

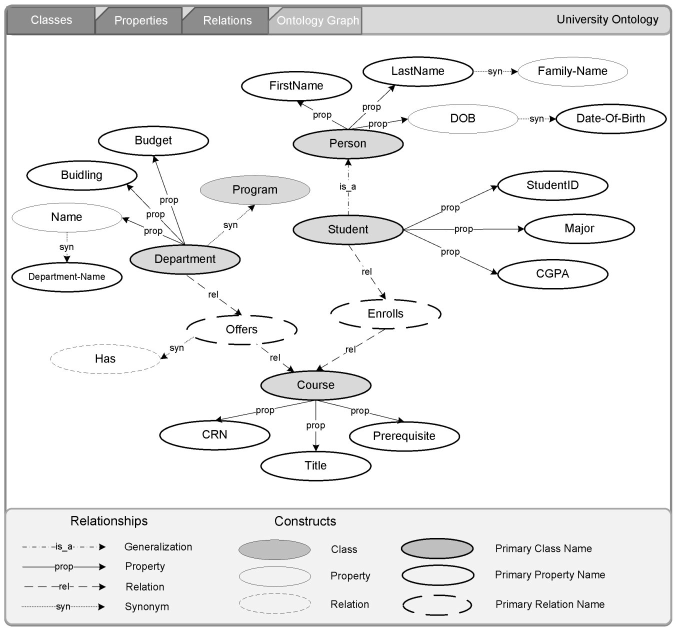


Fig. 15. Reference ontology graph

VI. Conclusion

We have introduced a novel approach to modeling and accessing information stored in legacy RDB systems. Its purpose is to provide the users of enterprise decision-support systems with direct, flexible, and customized access to information, through high-level semantic queries.

The approach is based on an innovative combination of multiagent systems (MAS) technology and Semantic Web (SW) technology, in which agents assist human partners in the development and maintenance of system ontologies, which in turn permits further delegation of operational tasks to agents.

Our proposed approach outlines an incremental evolutionary path that permits continuous operation of the system, requires no modification of the legacy databases, preserves organizational autonomy, and supports direct interaction between the decision-making user and the software system.

We have outlined a distributed multiagent architecture involving two types of subsystems: a database subsystem (DBS), comprising the relational database system and the server side of the agent-based middleware layer; and a user sybsystem (US), comprising the high-level customized user interface and the client side of the agent-based middleware. Multiple instances of US can be connected to a DBS, and a single US can interact with multiple DBS instances.

We have described how the reference ontology that defines the semantics of RDB structure is developed in interaction between the database administrator (DBA) and the database interface agent (DBIA). Initially, a base ontology is created automatically (using existing SW tools) from the RDB schema, and updated whenever the schema is modified. The agent endowed with a meta-ontology, representing the knowledge of the ontology-building process, interacts with the administrator to upgrade the base ontology into a full reference ontology, which is exported to the US side, in support of high-level semantic interaction with humans and agents seeking access to information through semantic queries.

The high-level user queries are expressed in a simplified natural language, using the concepts of reference ontology, as well as the concepts of user-specific custom ontology built for a specific user within a specific US instance; they are eventually translated to the base ontology level, then to SQL, and effected through on-demand conversion of data representation from the legacy system, without requiring any modification of its structure or content.

References

[1] C. Olszak and E. Ziemba, "Approach to Building and Implementing Business Intelligence Systems," *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 2, 2007.

[2] T. Berners-Lee, "Semantic Web Road Map," *W3C Design Issues Architectural and Philosophical Points*, 1998, retrieved May 03, 2010 from http://www.w3.org/DesignIssues/Semantic.html.

[3] M. Wooldridge, *An Introduction to Multiagent Systems*, 2nd ed. Glasgow, UK: Wiley, 2009.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. New Jersey, USA: Prentice Hall, 2003.

[5] 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*, vol. 19, pp. 643–670, 2009.

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

[7] G. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM*, vol. 38, pp. 39–41, 1995.

[8] M. Zubayer, "A multiagent architecture for semantic query access to legacy relational databases," Master's thesis, University of Northern British Columbia, Canada, 2011.

[9] N. Noy and D. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," Knowledge Systems, AI Laboratory, Stanford University, Tech. Rep. KSL-01-05, 2001.

[10] C. Bizer and R. Cyganiak, "D2RQ - Lessons Learned," *W3C Workshop on RDF Access to Relational Databases*, 2007. [Online]. Available: http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/w3c-d2rq-positionpaper/