Autonomous Evolution of Access to Information in Institutional Decision-Support Systems Using Agent and Semantic Web Technologies

Desanka Polajnar, Jernej Polajnar and Mohammad Zubayer

Abstract This chapter addresses the question of how an institutional decision-support system built on legacy relational databases (RDB) can evolve from a traditional database access model to a modern system that provides its decision-making users with agent-assisted direct semantic query access. We introduce a novel approach in which the system ontologies, developed autonomously within the institution, gradually co-evolve with the related ontologies accessible on the Web. This is achieved through cooperative developement of system ontologies by human domain specialists and software agents. The agents assist with ontology-building expertise, discovery of relevant knowledge on the Web, and ontology mediation. The underlying RDB need not be modified, which allows seamless transition and coexistence between access models. The approach is concretized as Semantic Query Access System (SQAS), a distributed system architecture based on agent-oriented middleware, in which database servers develop reference ontologies, while the application-oriented clients import and overlay them with user-specific custom ontologies.

1 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. As

D. Polajnar (✉) · J. Polajnar · M. Zubayer
University of Northern British Columbia, Prince George, BC V2N 4Z9, Canada
e-mail: desanka.polajnar@unbc.ca

J. Polajnar
e-mail: jernej.polajnar@unbc.ca

M. Zubayer
e-mail: zubayer@unbc.ca

F. Xhafa and N. Bessis (eds.), *Inter-cooperative Collective Intelligence: Techniques and Applications*, Studies in Computational Intelligence 495,
DOI: 10.1007/978-3-642-35016-0_6, © Springer-Verlag Berlin Heidelberg 2014

a consequence of global integration, changing market dynamics, and deployment of new technologies, institutional and corporate decision-support systems face increasingly steep requirements in regard to flexible, fast, and intelligent access to their underlying information repositories. 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. A major constraint in their efforts is the fact that vast amounts of relevant information, accumulated over time, reside in legacy systems that do not adequately support modern access techniques.

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) with traditional methods, non-technical users require technical assistance of database programmers, report writers, and application software developers, which involves delays, costs, and semantic gaps in human communication. In order to speed up access and give users more control, decision-support systems often rely on data warehousing techniques. Those techniques require information to be selected and extracted from operational databases, reorganized in terms of facts and dimensions, and stored in data warehouses [10]. 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 requires human mediation, time to restructure large amounts of data, and accurate foresight as to what information might be needed.

In this chapter, we explore an alternative approach, aimed at overcoming those limitations. It combines two fast-developing technologies—the Semantic Web (SW) [4] and multiagent systems (MAS) [16]—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 coexist with the more traditional ones. It allows continuous use of the legacy system. An important aspect of the new approach is that the transition from the traditional to the new access model can be effected gradually and autonomously within the host institution or company. This autonomy is significant because the resident domain expertise has an essential role in the transition, as well as because some of the relevant knowledge is often proprietary. A key premise underlying the approach is that the knowledge of the generic ontology-building process may be easier to standardize and formalize than the domain-specific knowledge particular to an organization and accumulated through work experience of its personnel. Accordingly, in the human-agent interactive and cooperative development of the institutional systems ontologies, the human partner should adopt the role of domain specialist and the agent the role of ontology-building specialist.

We present the approach in the form of an intelligent distributed system, called the Semantic Query Access System (SQAS) [11], 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. Differences stemming from the "closed world" [14] nature of enterprise systems (vs. "open world" SW) are also being studied (e.g., [12]). 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 cooperative design. 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 cooperative design. The approach relies on agents endowed with the technical knowledge of ontology-building procedures that assist human actors in the design process. A part of the agents' 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 chapter describes: the principles of our approach to agent-oriented semantic access in institutional decision-support systems (Sect. 2); the basic distributed architecture of SQAS (Sect. 3); the components constituting the intelligent middleware, their roles and interactions on the server (Sect. 4) and client (Sect. 5) subsystems, including agent-assisted development of reference and custom ontologies; a few closing remarks (Sect. 6); and the conclusions (Sect. 7).

2 Agent-Oriented Semantic Access

In this section we examine the basic requirements of user access to information stored in an existing relational database (RDB) in the context of an institutional or corporate decision support system. The user is a decision-making executive who formulates requests for information and receives reports from the system. Our model focuses on access and does not explicitly represent the various analytical processing that may be involved in report generation. The user is aware that the structure of the database and its contents may evolve over time. Our model assumes that such changes are introduced by the database administrator and does not represent the mechanisms by which they may be prompted or influenced by the

user. The user is familiar with the knowledge domain of the information in the database, but may differ in specific expertise and interests from other users of the same system. The user is not a database management specialist.

The requirements are developed in three steps. We first describe the requirements for a generic system that represents user access to information in an RDB in a way that is common to its many possible implementations. We then focus on the user-system interaction in legacy RDB systems. Finally, we examine user-system interaction through high-level semantic queries as represented in SQAS, and discuss its perceived practical advantages.

2.1 The generic system

The requirements for a generic system are shown in Fig. 1. They are described in terms of high-level 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 [13].

The actors of primary interest for us are User and 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. In particular, in the *Process Request* use case, the system accepts a request formulated by User, queries the database, and returns a report with the results formatted as requested. The *Manage Ontology* use case is concerned with bridging the semantic gap between User's domain-oriented terminology, often shaped by personal expertise and preferences, and the vocabulary of the database, whose meaning is specified in the database documentation, possibly with clarification of finer points provided by DBA. The last two use cases enable DBA and Data Entry Operator (DEO) to maintain the RDB structure and content respectively. In order to highlight the differences between legacy systems and SQAS, we next focus on two use cases, *Process Request* and *Manage Ontology*.

2.2 The Legacy RDB System

In a legacy RDB system, some of the generic system functions shown in Fig. 1 are performed on behalf of User by intermediary technical personnel, represented here by a human role called Report Writer; the rest 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.

In the *Process Request* use case, User explains to Report Writer what information should be retrieved and how it should be presented; Report Writer then

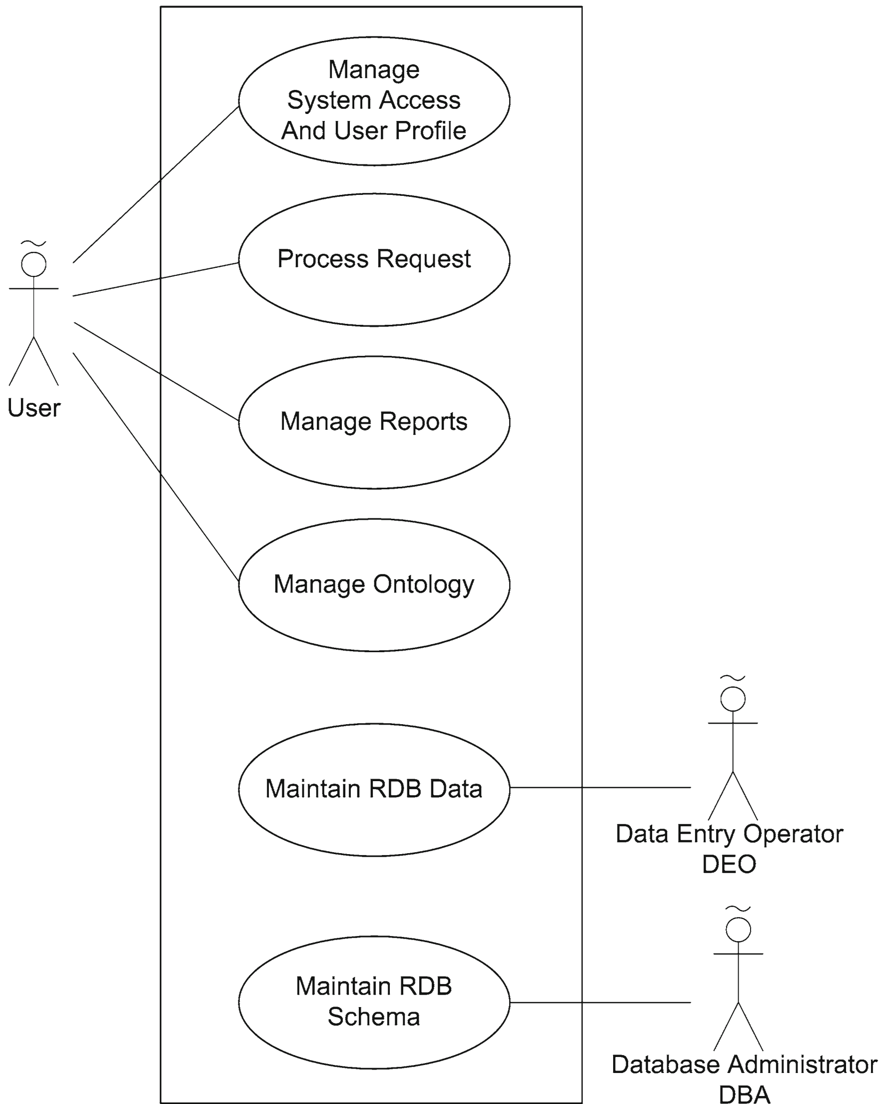

Fig. 1 The actors and high-level use cases of the generic system

queries the RDB to retrieve the information and presents it to User in the requested format. If the meaning of the request is not clear, Report Writer interacts with User in natural language in order to clarify it.

The *Manage Ontology* use case is concerned with the correspondence and translation between the database structures and their meaning, and the domain-oriented concepts and associated custom terminology employed by User. Apart from the basic relationships captured within the RDB schema, the meaning

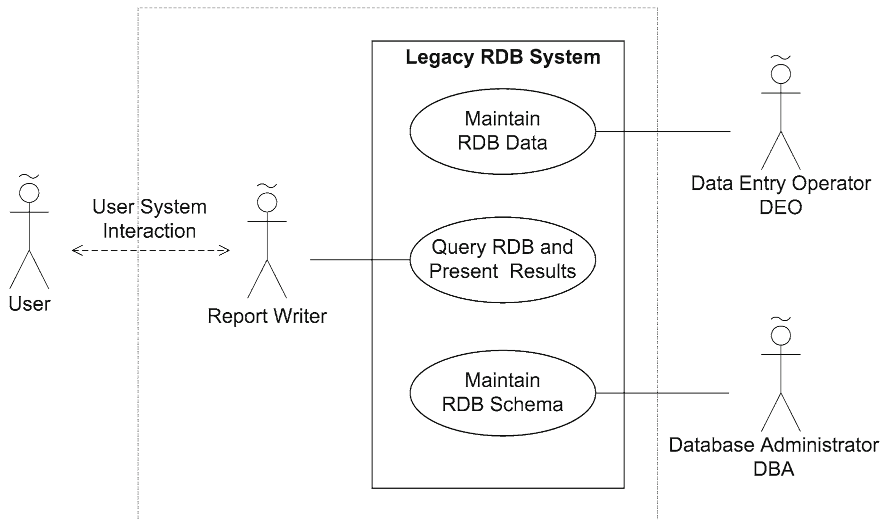

Fig. 2 User access to information in a legacy RDB system

of database structures is typically captured informally in natural-language documentation (and sometimes human knowledge) maintained by DBA. User's terminology and conceptual framework may differ from the ones presented by DBA. In order to learn both, Report Writer typically depends on informal documentation, and on natural-language interactions with both User and DBA. This process of consultation, negotiation, and delegation between User, Report Writer, and DBA is often time consuming and sometimes ambiguous, resulting in delays, costs, and occasional misunderstandings.

2.3 *The Semantic Query Access System*

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.

The *Process Request* use case allows User to directly communicate requests to the system, in a simplified natural language. In the request, User specifies what information should be retrieved and how it should be presented. If User's request is not clear, the system asks User for clarification of the request. This clarification process is an interactive one in which the system ensures that it understands User's request, similar to Report Writer in a legacy RDB system. It then retrieves the information and presents it in the requested format.

In the *Manage Ontology* use case, the meaning of database structures is formally captured in the *reference ontology*. The reference ontology represents the combined knowledge originating from the underlying RDB structure, the human actors in the system, and external ontologies available on the Semantic Web. 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 the RDB system. Similarly, User's conceptual framework and associated terminology are formally captured in the *custom ontology*. The custom ontology is a layer on top of the client's imported reference ontology; it is custom-built for each specific user. It is also built within the system itself, in interaction with 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.

2.4 Decomposition of SQAS Use Cases

The functions of each high-level use case can be further specified through decomposition into more elementary use cases. In presenting the decompositions of key generic use cases in SQAS, we also decompose the functionality into its client part, related to User, and its server part, related to the RDB. The client and server subsystems can reside on different machines and communicate through a network. In general, a client can interact with multiple servers, and a server with multiple clients; this is discussed in more detail in Sects. 3 and 6. For the moment, we consider the case of one client and one server.

The decomposition of the generic *Process Request* use case is shown in Fig. 3. In the client, the *Process SNL Request* use case allows User to formulate a request for information in Simplified Natural Language (SNL). The request contains domain-specific terms that describe the information to be retrieved, and keywords that describe the format in which it should be presented. Once the request is accepted, the *Parse SNL Request* use case produces an intermediate representation of the request, and the *Verify Request Semantics* use case checks that each statement as a whole in the request is semantically correct, including its use of custom ontology terms. If the SNL request is valid, the *Generate SPARQL Script* use case creates a SPARQL script from the intermediate representation of the request. The ontologies in SQAS are represented as Resource Description Framework (RDF) structures [6], and SPARQL [15] is the standard query language for RDF. The client then sends the SPARQL script to the server, and receives the SPARQL results from it. Finally, the SPARQL results are formatted and presented by the *Format and Display Report* use case.

In the server, the *Process SPARQL Request* 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 passes the SQL results to

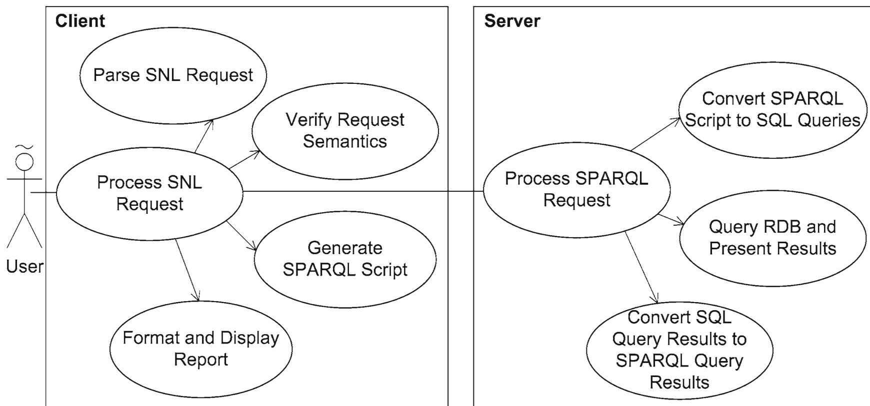

Fig. 3 The SQAS decomposition of the generic use case *Process request*

Convert SQL Query Results to SPARQL Query Results for translation. Finally, the *Process SPARQL Request* use case sends the results to the client. Note that the data remain permanently stored only in the RDB, and that RDF representations are created on demand as a request is processed. This approach does not require any modification in the RDB structure, and allows SQAS to coexist with other methods of accessing the legacy database.

The decomposition of the generic *Manage Ontology* use case is shown in Fig. 4. The use cases that primarily interact with User are assigned to the client subsystem, whereas the ones that primarily interact with DBA and RDB are assigned to the server subsystem.

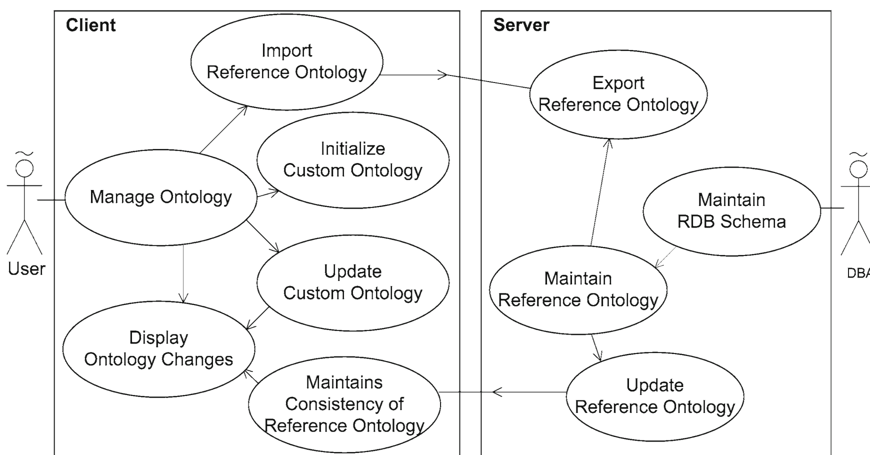

Fig. 4 Use case: *Manage ontology*

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. Thus the reference ontology is pulled by the client subsystem when it connects to the server initially, or reconnects following a period of disconnected operation. When changes occur in reference ontology while the client is connected, the updates are pushed to the client by the server. The reference ontology updates are displayed to the user by the *Display Ontology Changes* use case.

In the server subsystem, the *Export Reference Ontology* use case sends a copy of the reference ontology as requested by the newly attached client. The *Maintain RDB Schema* use case allows DBA to modify the structure of the RDB. When 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, which also pushes the updates to the attached client.

3 The SQAS Architecture

3.1 The High-Level Architecture of SQAS

At the high level, SQAS consists of any number of clients of the type *User Subsystem (US)* and any number of servers of the type *Database Subsystem (DBS)*. A single US can support multiple users. These subsystems can reside on different machines and communicate through a wide area network using a standard transport protocol. Figure 5 depicts a simple configuration, consisting of one single-user client subsystem and one server subsystem, that is used in most of the current presentation to explain the principles of system operation.

A US consists of an agent, called the *User Interface Agent (UIA)*, attached to each user, and a collection of interacting software components called the *User Interface Environment (UIE)*. Similarly, DBS consists of the *Database Interface Agent (DBIA)*, the *Database Interface Environment (DBIE)*, and the RDB system. The rest of this section outlines the agent roles of UIA and DBIA. Given the diversity of roles, each of these agents should preferably be internally designed as an agent team. Those internal designs and the associated issues of agent teamwork are beyond the scope of the current presentation.

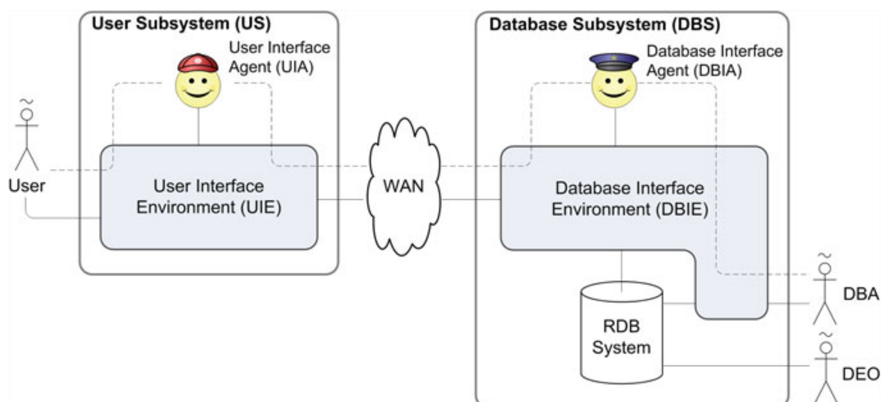

Fig. 5 A simple SQAS configuration with one single-user US and one DBS

3.2 The Roles of User Interface Agent

Assistance in SNL dialogue. User addresses the system in Simplified Natural Language (SNL). If the SNL processor generates a warning, UIA tries to autonomously resolve the issue in interaction with the subsystem components. If the SNL processor reports an error, UIA engages with User to correct it.

Searching the Semantic Web. The agent can search the Semantic Web for relevant external knowledge. For instance, it can look up synonyms and hypernyms of terms in natural language knowledge sources such as WordNet [9]. 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. UIA helps User create and maintain a custom ontology, a user-specific conceptual framework that is translatable to reference ontology. In order to maintain consistency, UIA ensures that any updates to the reference ontology are reflected in the custom ontology. UIA has the technical knowledge of the required ontology development and mediation mechanisms.

Customizing the behavior of User Interface Environment. While assisting User with SNL dialogue, UIA may learn from observations of User's preferential choices and customize the behavior of the user interface and possibly other UIE components, assuming that User elects to enable such options.

Coordination of reference ontologies. When US interacts with multiple DBSs, UIA acts to resolve any conflicts between imported reference ontologies.

3.3 The Roles of Database Interface Agent

Assistance in SNL dialogue. Apart from the conventional RDB operations on RDB, DBA addresses the system in SNL, with the agent's assistance as in US.

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

Development of reference ontology. DBIA interacts with DBA in developing and maintaining the reference ontology. A DBIE component called the *Schema to Base Ontology Mapper* analyzes the RDB schema and generates a Mapping File, which contains Resource Description Framework (RDF) models of the RDB schema. The Mapping File then serves as the *base ontology* from which DBA incrementally builds a full reference ontology with the assistance of the agent. DBIA provides technical guidance in the ontology development process, and accesses ontologies on the Semantic Web.

Customizing the behavior of Database Interface Environment. Primarily customizes the behavior of the user interface component, as in US.

4 Agent-Oriented Middleware for Server Subsystems

The software of SQAS is distributed between its client and server subsystems. 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 present discussion, 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 SQAS agents, with the users, and with the Semantic Web.

This section focuses on the internal structure of DBS, shown in Fig. 6. In Sect. 4.1 we discuss the DBIE components that support the main subsystem functions, in Sect. 4.2 a strategy for SQAS middleware implementation, and in Sect. 4.3 an example of an ontology-building scenario executed within DBS.

4.1 The Database Interface Environment

This environment comprises all software components that communicate with DBIA, DBA, DEO, and RDB. The solid lines represent direct communication between components. The dashed line represents communication between DBA and DBIA. A dashed envelope groups the components directly interacting with RDB that we revisit in Sect. 4.2. DBIE includes the following main components:

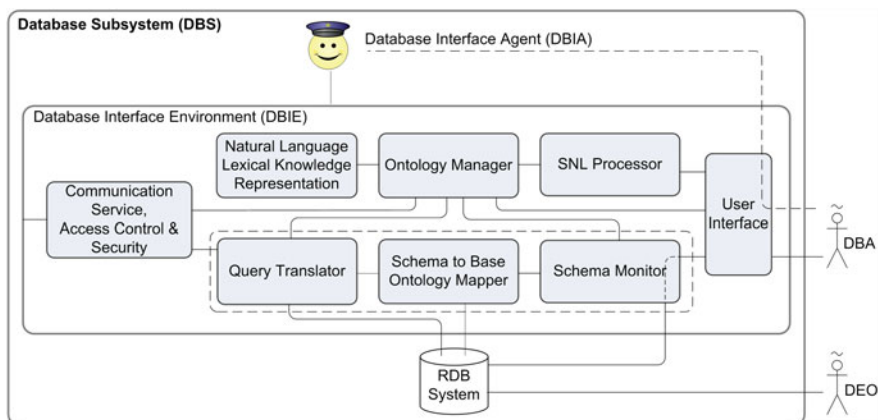

Fig. 6 The Database Subsystem

The *User Interface* provides an access point at which DBA interacts with DBS. Through it, DBA maintains the reference ontology with the assistance of DBIA and manages the RDB system. Based on its observation of user behavior and its learning abilities, the agent can intelligently adapt the interface to meet the user preferences.

The *SNL Processor* enables DBA to interact with the system using Simplified Natural Language (SNL), in addition to the more conventional user interface options. Given a user statement in SNL, the SNL processor first performs the lexical analysis and syntax analysis, using the SNL language definition, the vocabulary information from the reference ontology, and lexical information from the Natural Language Lexical Knowledge Representation component to generate an intermediate representation of the statement. After that, it performs semantic analysis, including ontology checking in interaction with the Ontology Manager, to verify that each statement as a whole is meaningful. Once the intermediate representation is generated and verified, the SNL Processor invokes the relevant components that execute the DBA's request. For instance, it can activate the Ontology Manager to update the reference ontology, or the agent to initiate a Web search.

The *Ontology Manager* is responsible for storage and maintenance of the reference ontology. 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 DBS are propagated to the instances of reference ontology in all participating USs. It also provides the SNL Processor the ontology information needed for semantic analysis.

The *Natural Language Lexical Knowledge Representation* component provides the meaning and semantic relations between natural-language concepts, as well as vocabulary knowledge, in both machine processable and human readable format. It

provides a language ontology which can be enhanced by acess to external language ontologies. The agent and the SNL Processor communicate with this component to look up meanings and relationships between natural language terms.

The *Query 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; finally, it converts the SQL query results to SPARQL query results. The Query Translator then sends the SPARQL results to the US.

The *Schema to Base Ontology Mapper* automatically generates a base ontology in the RDF format 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 Mapper re-generates the base ontology whenever it is alerted to a change in the RDB schema.

The *Schema Monitor* always listens for changes in the RDB schema made by DBA. When it detects a schema change it notifies Schema to Base Ontology Mapper to reflect the modifications in the base ontology, and then prompt the adjustments in the reference ontology.

The *Communication Service, Access Control, and Security* component facilitates all communications between US and DBS. By enforcing security features it ensures that no unauthorized access occurs.

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. The structural changes to RDB are introduced by DBA as modifications to the RDB schema, which are intercepted by the Schema Monitor and further result in modifications to the reference ontology.

4.2 A Note on Implementation Strategy

The intelligent midldleware of SQAS includes a variety of components, some of which would require both research and implementation efforts. The most innovative and research-oriented aspect of SQAS is the role of agents in ontology building. Many of the other components could be adapted from existing or future solutions in the development of Semantic Web, natural language processing, and some other areas. A plausible implementation strategy would be to adapt the architecture of SQAS as necessary in order to take full advantage of independently developed solutions and software components. In this section we briefly illustrate this approach with the three components of Database Interface Environment presented in the dashed envelope in Fig. 6.

The three compnents, namely the *Query Translator*, *Schema to Base Ontology Mapper*, and *Schema Monitor*, jointly provide the necessary conversions between the RDB schema and the base level of the reference ontology in RDF format, as

well as the actual translation of queries and results between the two formats. The functionalities of the first two components are provided by a number of existing tools, and most closely matched by the D2RQ platform [5]. 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 corresponds to an RDF representation of the RDB schema. In SQAS, the Mapping File represents the base ontology. The D2RQ Engine thus performs the role of Schema to Base Ontology Mapper. The Engine then uses the Mapping File to translate SPARQL queries to SQL queries, invokes the RDB, and translates the SQL results back to SPARQL results. The front end of the platform, the D2R Server, accepts SPARQL queries, passes them to the Engine, and presents the returned SPARQL results (RDF triples). The D2R Server and the D2RQ Engine thus match the role of the Translator component. The third component, the Schema Monitor, is a custom designed extension introduced in [17]. It is an interceptor component placed between DBA and RDB that recognizes the SQL commands which modify the RDB schema and prompts the D2RQ Engine to re-generate the Mapping File, i.e., the base ontology.

D2RQ can work with the Jade [3] agent platform with the assistance of Jena [8]. A Jade agent uses Jena's SPARQL capabilities for executing a SPARQL query on the D2RQ Platform.

4.3 A Scenario for Agent-Assisted Ontology Development

We will now have a closer look at the behavior of agents as ontology builders. 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. This includes the ability to understand the semantics of general ontological notions, such as class, subclass, property, and relationship. Such conceptual knowledge itself represents an ontology, to which we refer as *meta-ontology* (noting that this use of the term differs from its established meaning in philosophy). The agents must also have the procedural expertise in the development of knowledge representations. They provide technical guidance and assist their human partners in the construction of concrete ontologies for the knowledge domains specific to the given databases.

We illustrate the process with a few examples from a scenario in which a human designer (DBA) interacts with an agent (DBIA) to construct a reference ontology from a university relational database. The elements of the reference

ontology in these examples are constructed in Web Ontology Language (OWL). As a well-known ontology language, OWL is a convenient choice for presentation purposes. The choice is not intended to suggest that OWL representations are well suited for reasoning in SQAS, whose agents must deal both with the open world of the Semantic Web and with the closed world of the institutional information system. The questions related to the optimal choice of ontology language for SQAS are beyond the scope of this chapter.

As a first step in the development of reference ontology, its domain name is chosen, and a new name space is established, with a suitable prefix that allows one to differentiate between the names coming from different ontologies. Several types of names, each identified with a distinct standard prefix, may appear in the reference ontology. A concept may have several names, but those synonyms have different roles that are indicated by their standard prefixes.

A *base* name is introduced by the mapping of the RDB schema into base ontology. It is automatically derived from a term used in the RDB schema. For instance, from the RDB table name Department the mapper produces a base ontology entry map:Department a d2rq:ClassMap, which results in the base ontology class name entry <owl:Class rdf:ID=''bn.Department''/>, where the bn prefix identifies a base name. Base names of properties are constructed similarly. For instance, from the column name FirstName in Student table in the RDB, the mapper constructs map:Student_FirstName a d2rq:PropertyBridge; which results in the property base name FirstName of the base class Student in the reference ontology. The base names cannot be changed independently, because their role is to maintain the correspondence between the reference ontology and the underlying RDB schema.

When the mapping is completed, the agent presents each base name to the designer for the decision on the primary name of the same concept. The *primary* name of a concept is its unique official identifier within the reference ontology, as distinguished from other synonyms. The agents use the primary names (with prefix pn) in reasoning and interactions with environment components; the system allows user-specific synonyms (prefix un) in communication with human actors. The designer may adopt the base name as the primary name, or consider other choices. In the latter case, the agent may assist by offering natural language synonyms of the base name from a lexical knowledge source such as WordNet [9].

Once the base classes are defined, the designer and agent can define the more general classes. The superclasses can be defined in several ways. The designer may identify several existing classes that can be generalized into a new superclass, provide the primary name for the superclass, and let the agent create it. The agent may offer natural language synonyms before the name choice is finalized, or may look for natural language hypernyms that are common to the primary class names of all sunbclasses of the new class and offer them as candidate names for the new class. For example, the base classes Student and FacultyMember could be used to abstract a new superclass Person, for which there is no corresponding table in the RDB.

It should be noted that, since the entire reference ontology is ultimately derived from the RDB schema, the only concrete classes are the base classes; they are instantiated in the RDB, where all data reside. All other classes are abstract. This distinction influences the handling of property names in superclasses. If a superclass property is designated by identical primary names in all of its subclasses, and all the subclasses are abstract, the entries for that property can be removed from the subclass descriptions, as the property will be inherited from the superclass once it has been defined. However, base classes retain their properties with their associated base names in order to maintain the translatability to the RDB schema. Another observation is that when introducing new superclasses in the case where the primary names of the corresponding properties in subclasses do not match, the designer may be tempted to revisit the subclass definitions and rename the properties in order to remove the name conflicts. This may be simple in a very early design stage when the reference ontology has not been exported to client systems and the dependencies in the existing local software are few and easily traceable; later on, a change of primary property name in an existing class may require a lot of maintainance in derived ontologies and applications, and reliance on synonym management may be preferable to a primary name change. Again, agent's assistance and ability to track the implications of a potential change may be highly valuable to the designer.

The Schema to Base Ontology Mapper can recognize relations between RDB tables that result in class relations within the reference ontology. For instance, the following map entry identifes the ontology relation in which a university department offers a course:

```
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";
```

When the building of reference ontology is completed, the agent dispays it as an editable graph to the designer for modification and approval (Fig. 7).

5 Agent-Oriented Middleware for Client Subsystems

The architectural structure of the US is shown in Fig. 8. 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

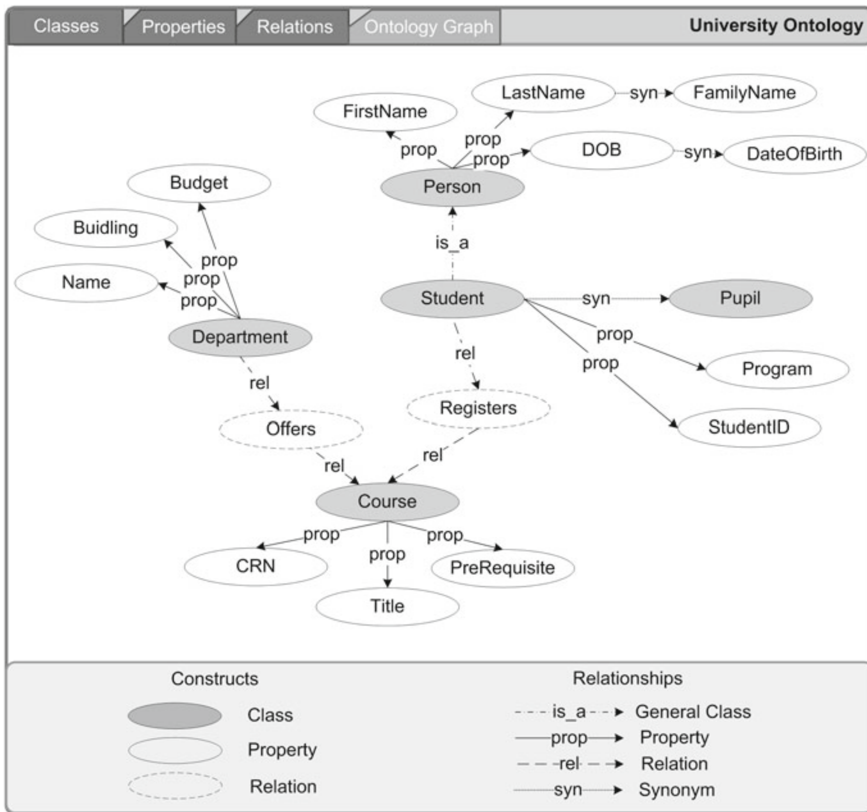

Fig. 7 The reference ontology graph for a small university RDB

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 its objectives without additional human assistance.

5.1 The 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 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:

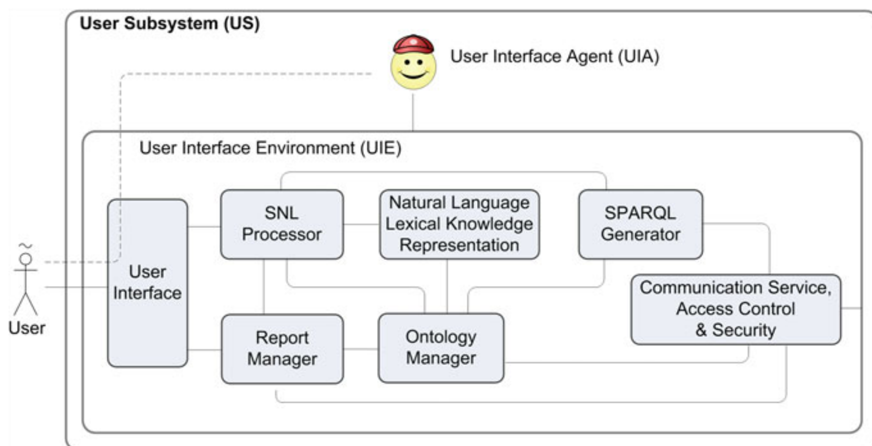

Fig. 8 The User Subsystem

The *User Interface (UI)* enables all communications between the user and the system. It provides the system access functionality as formulated in Sect. 2.3.

The *SNL Processor* component enables the user to interact with the system using SNL. The analysis of input statements is similar as in DBS, except that the semantic verification consults the custom ontology, and through it the reference ontology. Once the intermediate representation is successfully generated and verified, the SNL Processor invokes the relevant components and passes to them the relevant parts of the intermediate representation. If the statement is a request for information, the SNL Processor invokes the SPARQL Generator with the query-related information, and the Report Manager with the formatting instructions. Otherwise, it forwards the statements to the Ontology Manager.

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 User.

The *Natural Language Lexical Knowledge Representation* component has an identical role as in DBS. 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 user authentication, privileges, security, and the interactions with lower-layer communication services.

5.2 A Scenario for Agent-Assisted Semantic Access

We can now illustrate how a user accesses information in SQAS with a simple scenario in which a request is entered and a report is generated. This scenario assumes that the user requests information stored in the RDB through the User Subsystem (US), which has a copy of the reference ontology consistent with the original in the DBS. The US also has a custom ontology for the current user, which defines user-specific concepts using constructs in the reference ontology.

The user formulates a request for report in the Simplified Natural Language (SNL) and submits it through the User Interface (UI):

```
Generate list of students that registered
   for Fall 2012.
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.
```

Some of the words in an SNL request express the control structure and other relationships in simple English (e.g., that, with, using); others directly relate to system actions (generate, format); and some have a defined meaning in the custom or reference ontologies (student, last name). In all three categories the user has the flexibility of defining custom terms. In this scenario, we only consider the translation of custom ontology terms.

The current request has three related 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 first two statements constitute a query, and the last specifies the report generation.

The SNL Processor analyzes the request online, allowing the agent and the user to deal with any arising problems. The Processor first performs lexical analysis in which it breaks the SNL text into a sequence of tokens, such as words and symbols. This is followed by syntax analysis that checks whether the text is grammatically correct and generates an initial intermediate representation. Next, the Processor performs semantic analysis to determine if the statements in the request

are meaningful. In this step it may consult the language definition, as well as the custom and reference ontologies. In particular, this implies that each ontological term is ultimately translatable to the base ontology level (which implies that it meaningfully relates to the RDB schema). During this step, the Processor recognizes the actions that need to be performed and verifies whether all parameters that are needed for these actions are present. The fully verified intermediate representation is further transformed so that it can be executed by invoking the appropriate components in the environment.

In each of the described steps, the SNL processor may produce a warning or an error. For example, during the semantic analysis, the SNL processor searches both the natural language vocabulary in the lexical knowledge component and the custom and reference ontologies, to determine whether the word or phrase has a purely natural language meaning or a technical meaning. If both searches are successful, there may be an ambiguity to resolve. For example, the words *First* and *Name* both have general meaning in English, but the expression *First Name* is found as a property of the class *Student* in the reference ontology and hence has a technical meaning. The warning raised by the SNL Processor in situations of this type would be intercepted by the agent that would usually resolve it in favor of the technical interpretation autonomously, without invoking the option of consulting the user.

Another situation that prompts agent's intervention arises when a term, such as `registered` in our current example, appears from the context to have a technical meaning, but cannot be found in that exact form. The agent may look for lexically similar terms and find that there is a class called `registration`. The user may be consulted for clarification and also prompted to add the definition of the term `registered` to the custom ontology.

An important aspect of SNL processing is that all custom ontology terms in the query must be eliminated and replaced by reference ontology terms. The intermediate representation of the query contained in the request will be passed to SPARQL Generator, and the SPARQL script that it produces will proceed for further processing to DBS, where the custom ontology is not known. Since custom ontology terms are more likely to be involved in ambiguity resolution that requires linguistic analysis and occasional interventions by the agent and user, the SNL Processor is the most suitable component to effect the custom term elimination with the assistance of Ontology Manager. The immediate representation links the reference terms to their ontology definitions. It is left to SPARQL Generator, as its preprocessing step, to translate the reference ontology terms into base ontology terms, which are then used to generate the SPARQL script. In our current example, the primary name `date of birth` is replaced by the base name `DOB`. As a part of the preprocessing, SPARQL Generator removes the leading tag (`bn.`) from each base term.

The SNL Processor executes the commmands, such as *Generate*, *Include*, and *Format*, by structuring the intermediate representation and passing its relevant parts to specific components in the environment. When executing the request in our current example, the SNL Processor forwards the intermediate representation

of the query derived from the first two statements to SPARQL Generator, and the formatting information derived from all three statements to Report Manager. The SPARQL Generator then translates the query information to SPARQL script and sends it on to DBS, while Report Manager uses the information to produce the formatted report when SPARQL query results come back from DBS. Note that it is not necessary to purge the custom terms from the information passed by the SNL Processor to Report Manager, as the report may indeed employ some user-specific terminology involved in the formulation of the request.

The SPARQL Generator constructs a SPARQL script from the commands and parameters received from the SNL Processor. In general, the script may contain multiple SPARQL queries; in the current example there is only one. The SPARQL Generator divides the technical terms in the first statement into two categories: the 'basic terms' set, consisting of terms appearing before the keyword *that*, and 'conditional terms' set consisting of the terms that follow. This distinction later helps the generator in constructing the script. A SPARQL query is made up of three components: the PREFIX declaration, the SELECT clause, and the WHERE clause. The generator gets the base URI `base=''http://localhost:2020/vocab/resource/''` from the reference ontology header and includes it in the query as a PREFIX. In the prefix declaration, it replaces the equals symbol (=) with a colon (`:`), and the quotes with opening (<) and closing (>) tags. 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. Those variables are taken from the technical terms 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 our current example, the variables are `?StudentID`, `?FirstName`, `?LastName`, `?DOB`, and `?CGPA`.

In the WHERE clause, a number of triples are constructed. A triple 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 terms' set (`student`). The predicate is a technical word in the URI format (`PREFIX:Class_Property`), constructed in two steps. First, SPARQL Generator concatenates a class name from the 'basic terms' set and a property name from the SELECT clause with an underscore symbol (_). Second, it concatenates the prefix (`base`) and the previously created segment (`Class_Property`) with a colon symbol (`:`). The object variable is constructed using the property name. Following this method the generator constructs a triple for each variable appearing in the SELECT clause. Finally, the generator constructs a triple for each property name from the 'conditional terms' set (`Semester`, `Year`), using the class name variable from the 'conditional terms' set (`?registration`) as subject, and the specified property value as object. These two groups of triples are then linked with a third triple whose predicate has the property `StudentID`, which is a common property between the class in the 'basic terms' set and the class in the 'conditional terms' set.

The complete SPARQL script for our current example is:

```
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 "2012".
}
```

Once the SPARQL script is constructed, the Communication Service, Access Control and Security component sends it to the destination DBS. The corresponding component in 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 Translator generates the equivalent SQL queries. The SQL query generated in our example is:

```
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 = '2012'
```

The D2RQ Engine executes the SQL queries on the RDB system and retrieves SQL results. Query Translator then converts them from SQL format to SPARQL format and Communication sends them to the US. A subset of the generated SPARQL results is shown in Fig. 9.

The Report Manager component in US receives the SPARQL results from DBS. It then formats the results according to the instructions provided in the request by the user. A user selected template (format-k) is used for displaying the report. It also sorts the SPARQL results alphabetically by `LastName`. Report Manager refers to Ontology Manager to replace any base name with its primary name or user-specific name. It then displays the formatted report to the user. The report generated from the SPARQL results is shown in Fig. 10.

Fig. 9 The SPARQL results

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

. . .

6 Closing Remarks

A few points remain to be made on issues that become apparent when envisioning a full-scale architecture of a system such as SQAS. These issues have been indicated but not elaborated while we presented the main principles in a simplified setting.

In a large and complex information system, a client subsystem would typically not import the complete reference ontology that a server has developed, but only its part, or view, that is relevant to the users of the particular client subsystem. This requires methods for specifying views and for their coordinated maintenance by the Ontology Manager components of the client and server.

A client subsystem would in general connect to multiple servers and form its reference ontology by composing the views imported from them. Therefore each entry in the client subsystem's reference ontology must carry a tag indicating its source, which enables SPARQL Generator to produce separate scripts for queries directed to different servers, and Report Generator to correctly integrate the retrieved information. While the basic mechanisms require little adjustment, the objective of smooth and user-friendly integration of views increases the complexity of ontology management and involves various research questions.

When a group of users naturally share a common composite view of reference ontology, the designer may choose to give each user a separate replica or let them share access to the same copy. In the latter case, individual users may still need distinct custom ontologies, which increases the complexity of ontology management.

Even a brief enumeration of developmental issues indicates the centrality of ontology management tasks to the semantic access framework formulated above. Many of the arising questions appear to belong to mainstream research topics motivated by a wide variety of potential applications. There are strong indications that the resulting ontology management techniques will require careful balancing of conflicting objectives, reasoning, autonomous judgment, learning from experience, and intelligent interaction with other entities. The development of such techniques along the lines proposed in this chapter could find immediate

Registered Students
Date: January 10, 2012

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
...				

Fig. 10 The formatted report

application in the processing layer of context aware systems [2], in acquisitional query processing systems [7], and in a number of other rapidly advancing areas. This reinforces our view that agents will play a significant role in ontology building and management in complex intelligent systems of the future.

7 Conclusions

This chapter makes the case for the use of intelligent agents in ontology-building tasks as a means for autonomous evolution of conventional decision-support systems in institutional or corporate environments towards modern systems that provide flexible and direct access to information through high-level semantic queries. This novel approach outlines an incremental evolutionary path that permits continuous operation of the system, requires no modification of the legacy databases and allows conventional access to them, preserves organizational autonomy, and supports direct semantic-query interactions between the decision-making user and the software system, without intervening personnel.

The approach is based on an innovative combination of multiagent systems and Semantic Web technologies, 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. The envisioned system has a distributed architecture with any number of client and server subsystems connected by a wide area network and functionally integrated through a layer of agent-oriented middleware. A server contans a legacy relational database along with a reference ontology, autonomously developed within the system through human-agent

cooperation, that represents the semantics of the database schema. This allows online translation of SPARQL queries into SQL queries and conversion of retrieved SQL results back into SPARQL format. A client subsystem allows its users to formulate semantic queries in a simplified natural language, using high-level terms from the composite reference ontology whose component views are imported from servers, as well as user-specific terms from a custom ontology that is co-developed by the user and an agent as a layer on top of the reference ontology. A semantic query is translated into SPARQL scripts for distributed execution on appropriate severs. The client integrates and delivers the retrieved results. The co-development of reference ontology and the execution of a semantic query are ilustrated by typical scenarios.

The approach suggests that significant practical benefits could result from endowing agents and agent teams with ontology management capabilities. Many of the necessary preconditions for this, both in terms of formal understanding and modeling of ontology management processes and also in terms of available software tools for development, mediation, and maintenance of ontologies, are either already appearing or likely to to be brought about by research in Semantic Web technologies and applications. The motivation, feasibility, and potential benefits of agent-oriented ontology management applications are likely to be enhanced by increased availability of public knowledge resources that agents could access. These considerations motivate further studies in meta-ontologies and techniques for agent-oriented ontology management. Possible directions of research into further applications of the current approach include context-aware systems, acquisitional query processing systems, and other rapidly advancing areas.

References

1. G. Antoniou, F. Harmelen, Web ontology language: Owl, in: Handbook on Ontologies, International Handbooks on Information Systems, edited by S. Staab, R. Studer (Springer, Berlin Heidelberg 2009), pp. 91–110
2. M. Baldauf, S. Dustdar, F. Rosenberg, A survey on context-aware systems. Int. J. Ad Hoc Ubiquitous Comput. **2**(4), 263–277 (2007)
3. F. Bellifemine, G. Caire, D. Greenwood, *Developing Multi-Agent Systems with JADE* (Wiley, Wiltshire, 2007)
4. 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
5. Bizer, C., Seaborne, A.: D2RQ-Treating non-RDF Databases as Virtual RDF Graphs. In: Proceedings of the 3rd International Semantic Web Conference (ISWC2004). Hiroshima (2004).
6. D. Brickley, R. Guha, RDF Vocabulary Description Language 1.0: RDF Schema. Tech. rep., W3C (2004). Retrieved October 06, 2010 from http://www.w3.org/TR/2004/REC-rdf-schema-20040210/
7. S.R. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TinyDB: An acquisitional query processing system for sensor networks. ACM Trans. Database Syst. **30**(1), 122–173 (2005)

8. B. McBride, D. Boothby, 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
9. G. Miller, WordNet: a lexical database for English. Comm. ACM **38**, 39–41 (1995)
10. C. Olszak, E. Ziemba, Approach to building and implementing business intelligence systems. Interdisc. J. Inf., Knowl., Manage. **2**,134–148 (2007)
11. D. Polajnar, M. Zubayer, J. Polajnar, A multiagent architecture for semantic access to legacy relational databases. In: 2012 IEEE International Systems Conference (SysCon), pp. 1–8 (2012). doi 10.1109/SysCon.2012.6189521.
12. F. Ricca, L. Gallucci, R. Schindlauer, T. Dell'Armi, G. Grasso, N. Leone, OntoDLV: an ASP-based system for enterprise ontologies. J. Logic Comput. **19**, 643–670 (2009)
13. J. Rumbaugh, I. Jacobson, G. Booch, *Unified Modeling Language Reference Manual*, 2nd edn. (Pearson, Higher Education, 2004)
14. S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd edn. (Prentice Hall, New Jersey, 2003)
15. W3C: SPARQL Query Language for RDF (2008). Retrieved January 18, 2013 from http://www.w3.org/TR/rdf-sparql-query/
16. M. Wooldridge, *An Introduction to Multiagent Systems*, 2nd edn. (Wiley, Glasgow, 2009)
17. M. Zubayer, *A Multiagent Architecture for Semantic Query Access to Legacy Relational Databases* (University of Northern British Columbia, Canada, 2011). Master's thesis