Transparent Access to Relational, Autonomous and Distributed Databases Using Semantic Web and Service Oriented Technologies

Bruno José de Sales Caires

(Licenciado)

Tese Submetida à Universidade da Madeira para a

Obtenção do Grau de Mestre em Engenharia Informática

Funchal – Portugal

Setembro 2007

Supervisor:

Professor Doutor António Jorge Silva Cardoso

Professor Auxiliar do Departamento de Matemática e Engenharias da Universidade da Madeira

ABSTRACT

With the constant grow of enterprises and the need to share information across departments and business areas becomes more critical, companies are turning to integration to provide a method for interconnecting heterogeneous, distributed and autonomous systems. Whether the sales application needs to interface with the inventory application, the procurement application connect to an auction site, it seems that any application can be made better by integrating it with other applications.

Integration between applications can face several troublesome due the fact that applications may not have been designed and implemented having integration in mind. Regarding to integration issues, two tier software systems, composed by the database tier and by the "front-end" tier (interface), have shown some limitations. As a solution to overcome the two tier limitations, three tier systems were proposed in the literature. Thus, by adding a middle-tier (referred as middleware) between the database tier and the "front-end" tier (or simply referred application), three main benefits emerge. The first benefit is related with the fact that the division of software systems in three tiers enables increased integration capabilities with other systems. The second benefit is related with the fact that any modifications to the individual tiers may be carried out without necessarily affecting the other tiers and integrated systems and the third benefit, consequence of the others, is related with less maintenance tasks in software system and in all integrated systems.

Concerning software development in three tiers, this dissertation focus on two emerging technologies, Semantic Web and Service Oriented Architecture, combined with middleware. These two technologies blended with middleware, which resulted in the development of Swoat framework (Service and Semantic Web Oriented ArchiTecture), lead to the following four synergic advantages: (1) allow the creation of loosely-coupled systems, decoupling the database from "front-end" tiers, therefore reducing maintenance; (2) the database schema is transparent to "front-end" tiers which are aware of the information model (or domain model) that describes what data is accessible; (3) integration with other heterogeneous systems is allowed by providing services provided by the middleware; (4) the service request by the "front-end" tier focus on 'what' data and not on 'where' and 'how' related issues, reducing this way the application development time by developers.

KEYWORDS

Data / Information Integration

Semantic Web

Ontology

Web Services

Middleware

RESUMO

Com o crescimento das organizações e com a necessidade de partilha de informação entre departamentos e área de negócio a ser um factor crítico, as organizações estão a recorrer à integração como forma de interligar sistemas heterogéneos, distribuídos e autónomos. Por exemplo, seja o sistema de vendas que necessite de interagir com o sistema de inventário, o sistema de aquisições que necessite de ser ligado com o site de compra de produtos, torna-se óbvio que qualquer aplicação pode-se tornar mais útil ao ser integrada com outras aplicações.

A integração entre aplicações pode encontrar várias barreiras devido ao facto destas não terem sido desenhadas e implementadas tendo a possibilidade de integração como requisito. Relacionado com questões de integração, os sistemas de duas camadas, compostos por uma camada de base de dados e por uma camada de interface ("front-end"), têm evidenciado algumas limitações. Como solução para ultrapassar as limitações próprias dos sistemas de duas camadas, os sistemas de três camadas foram propostos na literatura. Assim, ao adicionar uma camada intermédia (referida por middleware) entre a camada de base de dados e a camada de interface (ou simplesmente referida por aplicação) surgem três benefícios principais. O primeiro benefício está relacionado com o facto da divisão do software em três camadas aumentar as capacidades de integração com outros sistemas. O segundo benefício relaciona-se com o facto de poderem ocorrer modificações em cada camada individual sem ter necessariamente que afectar as outras camadas e sistemas integrados. O terceiro benefício, consequência dos outros dois, reflecte-se nas reduzidas tarefas de manutenção nos sistemas de software e em todos os sistemas integrados.

Relacionada com o desenvolvimento de software em três camadas, esta dissertação foca-se em duas tecnologias recentes, Semântica Web e arquitecturas orientadas a serviços, combinadas com o middleware (camada intermédia). Estas duas tecnologias quando utilizadas em conjunto com a camada intermédia, que resultou no desenvolvimento da "framework Swoat" (Service and Semantic Web Oriented ArchiTecture), conduz ao aparecimento das seguintes quatro vantagens: (1) permite a criação de sistemas soltos (loosely coupled), desacoplando as camadas de a base de dados das camadas de interface reduzindo desta forma a manutenção; (2) o esquema da base de dados é transparente para as camadas de interface que estão dependentes

do modelo de informação (ou modelo de domínio) que descreve quais os dados que estão acessíveis; (3) a integração com outros sistemas heterogéneos é permitida através do fornecimento de serviços pela camada intermédia; (4) a invocação de serviços feita pela camada de interface foca-se em descrever 'que dados' e não em questões relacionadas com o 'onde' e 'porquê', reduzindo desta forma o tempo de desenvolvimento das aplicações pelos programadores.

PALAVRAS CHAVE

Integração de Informação / Dados

Middleware

Ontologias

Semantic Web

Serviços Web

In memory of my grandmother Filomena Sales.

ACKNOWLEDGMENTS

Special acknowledgment to Professor Jorge Cardoso that accepted to be the supervisor. Regards for the great guidance, for the motivation provided in the most difficult dissertation stages and for teaching me how to research.

Regards to the "University of Madeira" computer department team for sharing their experience during the excellent three years which I spent collaborating with the team. Regards to Dr. António Pires , Eng. Duarte Costa, Eng. Gonçalo Sol, Eng. Paulo Silva, Eng. Pedro Valente which I had the pleasure to work with.

Also, regards to Eng. Rui Alves – SkySoft - for the several discussion lunches which finished with a paper napkin full of ideas and suggestions.

Regards to Eng. Miguel Gouveia – Secretaria Regional de Educação and SEED– for the fruitful coffee discussions and ideas. Thanks for your unselfish support.

Regards to Eng. Tiago Silva – Serviço Regional de Saúde, E.P.E. – for the great ideas and suggestions. Also, thanks to Dr. Carlos Rebolo and Dr. Carlos Martins– Serviço Regional de Saúde, E.P.E. – for the motivation provided in order to finish the dissertation.

To may mother, father, brothers and sisters-in-law for being present and for helping me to overcome the difficult bad mood stages.

To "Clube de Aventura da Madeira" for great the trekkings in madeira island.

To all my friends that provided motivation and encouragement, one special regard.

TABLE OF CONTENTS

LIST OF FIGURES

ACRONYMS

API - Application Programming Interface

ASP - Active Server Page(s)

BS - Business Services

COG - Corporate Ontology Grid

COTS - Commercial Off-The-Shelf

DI - Data Integration

DS - Data Services

EAI - Enterprise Application Integration

EAS - Enterprise Applications Systems

EDS - Enterprise Data Sources

EII - Enterprise Information Integration

EPS - Enterprise Process Systems

ETL - Extraction, Transformation and Loading

FOL - First Order Logic

FS - Functional Services

FSS - Federated Schema Service

GAV - Global-as-View

GUI - Graphical User Interface

GVV - Global Virtual View

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

IBHIS - Integration Broker for Heterogeneous Information Sources

IT - Information Technology

KIF - Knowledge Interchange Format

KRAFT - Knowledge Reuse and Fusion/Transformation

LAV - Local-as-View

MOMIS - Mediator envirOnment for Multiple Information Sources

ODSOI - Ontology-Driven Service-Oriented Integration

OWL - Web Ontology Language

PHP - Hypertext Preprocessor (HTML-embedded scripting language)

RDBMS - Relational Database Management System

RDF - Resource Description Framework

RDQL - RDF Data Query Language

RQL - RDF Query Language

SeRQL - Sesame RDF Query Language

SOA - Service Oriented Architecture

SOAP - Simple Object Access Protocol

SPARQL - Simple Protocol and RDF Query Language

SQL - Structured Query Language

Swoat - Service and Semantic Web Oriented ArchiTecture

SWT - Semantic Web Technologies

UDDI - Universal Description, Discovery and Integration

URI - Universal Resource Identifier

URN - Uniform Resource Name

W3C - World Wide Web Consortium

WS - Web Services

WSDL - Web Services Description Language

WWW - World Wide Web

XML - Extensible Markup Language

1. INTRODUCTION

> "The ideal engineer is a composite ... He is not a scientist, he is not a mathematician, he is not a sociologist or a writer; but he may use the knowledge and techniques of any or all of these disciplines in solving engineering problems."
> —N. W. Dougherty (1955)

Many software solutions deployed in organizations are composed by two main tiers: database (used to store the data) and the "front-end" (the interface used for user interaction). For example, it is expected that an organization that has two systems, for example, the Human Resource Management system and the Account system, also has two databases (the account and the human resource) and two "front-end" tiers (one per system). When analysing the software systems of organizations, essentially in small organizations, it is possible to conclude that software systems usually follow a two tier approach, and more specifically, the existent databases in the organization are usually relational and the "front-end" tiers are Web based or GUI (following the windows paradigm).

In the above mentioned situation, the existence of several software systems in the organization leads to islands of information scattered by several organization functional areas (like finance, accounting, etc). Consequently, when two or more software systems exist in the organization, there is a need to integrate them (or establish connections between them) because often software systems, mainly through "front-end" tiers, need to access data from several databases. This way, it is expected

that, for example, the Account system "front-end" tier also access the Human Resource database system because the employee's data is used by both software systems. This way, the Account systems users, using the "front-tier" will manipulate data stored in several databases.

Regarding to development of "front-end" tiers that need to be integrated with one or more databases, two main challenges are faced by engineers and software mergers. The first challenge is related with the creation of *loosely coupled* systems (decoupled systems). Decoupling means that two integrated or connected systems should know the less possible about the each other. This way, the main purpose is that when changes occur in the database, for example deleting a database table, its propagation be minimized, therefore reducing the changes in all connected "front-end" tiers. The second issue is related with *reusing* already implemented functions in heterogeneous systems. For example, suppose that two "front-end" tiers (applications) are connected with the human resource database: one is Web based and the other is GUI. The functions developed to manipulate the data by the Web based applications should also be reused by the GUI application.

In order to address the above mentioned topics, this dissertation explores the use of Semantic Web technologies (SWT) and Service Oriented Architectures (SOA), combined with middleware (deployed between the database and the "front-end") to provide a possible solution to the presented problems.

This introduction chapter starts by presenting the scenario that motivated this dissertation development. It follows with the problem statement, detailing the above mentioned high level problems. The dissertation objectives and the approach to solve the problem are also presented, finalizing with the dissertation organization.

1.1. MOTIVATION SCENARIO

This motivation scenario presents an organization that owns several software systems, either developed internally or externally (referred Commercial-Of-The-Self: COTS). In this scenario all software systems store their data on relational databases.

Referring to examples of COTS systems deployed in the organization, the Human Resource Management system and the Accounting system are just two of these.

Referring to the systems developed in-house, that refers to software systems related with the organization core business (related with the main organization purpose), on characteristic of these systems is that they use data stored in several other database systems (like the Human Resource Management system and the Accounting system). This characteristic is explained by the fact that these systems refer to the main organization purpose, called organization core business, which is the most important, therefore requiring data from all the other systems.

For example, instantiating this scenario in a health organization, the core business systems are related with the clinical area. All the other systems that support the organization like the Human Resource system and the Account system among others are not core business and therefore COTS. These last are not developed using internal resources but instead bought from an external organization. In this case, the clinical area systems need to access to the Human Resource systems in order to allow or deny systems access.

Detailing the organization core business systems, two types of clients (also referred "front-end" tiers) are found in the organization: GUI and Web based (Berry, 2005) clients. In this scenario, GUI applications are typically developed in Java and used by organization employees in order to perform specific tasks (like insert the personal data in the human resource management system). On the other side, Web applications are typically developed using languages such as PHP, ASP, etc. These are used in specific cases in other to allow access to applications for authenticated users, either internally or externally from the organization.

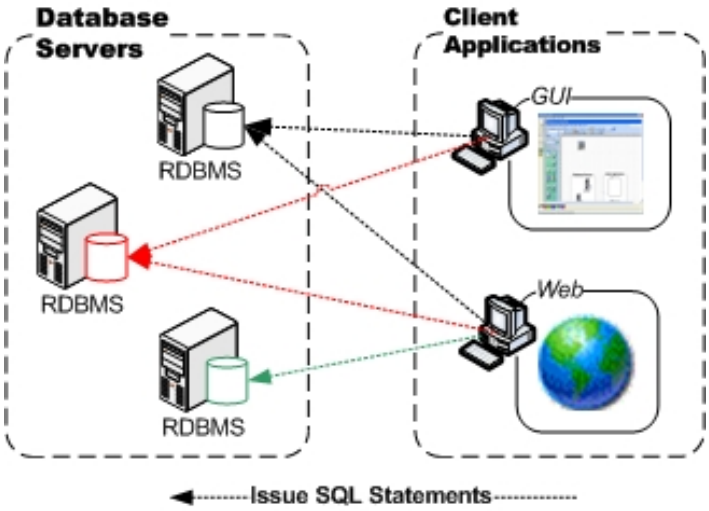


Figure 1 - Connections between clients and relational databases

One characteristic of this scenario is that usually client applications, illustrated at the right part of Figure 1, are not only responsible for allowing user interaction but also for extracting data from several databases, therefore integrating it. This way, and as illustrated in Figure 1, clients (right part of the figure) are directly connected to databases (left part of the figure) and one application (Web based and GUI) can be connected to one or more databases.

The following section details the problems that emerge from this scenario.

1.2. PROBLEM STATEMENT

The existence of several systems, frequently composed by a database and a "front-end", usually leads to integration needs. The integration with other systems is, most of the times, achieved by connecting the application ("front-end" tier) to one or more databases (usually one is the application database and the others are of other applications). The presented scenario can lead to an assortment of disadvantages:

- The "front-end" application is dependent of the database technology. This dependency can be related with the paradigm of the database (relational, object oriented, XML, etc) and also with the software product used (MySql, Sql Server, Oracle, etc). In this case, when these types of changes occur on the database, all the applications connected to the database will have to be changed.

- The "front-end" application is vulnerable to changes on the database tables. Due to new requirements of the system, usually the database has to be changed in order to respond to the new demands. Therefore, new tables are added, deleted or changed. This case can also lead to changes on connected clients.

- Two tier architectures (the described scenario) have disadvantages related with the lack of reusability of the functions implemented. For example, suppose a database that has two connected "front-end" clients: one developed in Java (Graphical User Interface) and other developed in PHP (to be accessed via Web). The functions developed in order to be used in the Java application should be reused by the application developed in Web. This means that the application functions should be accessible across heterogeneous systems and not only by the application in which the functions are implemented. For example, it may be desirable that all the functions developed to the human resource application be reused by other applications that need to integrate with the human resource application.

In order to provide a possible solution to this problem, this dissertation explores the use of Semantic Web Technologies and Services Oriented Architectures combined with middleware, deployed between database tier and "front-end" tier. Therefore, the

proposed solution adds one extra layer between the database tier and the "front-end" tier.

The next section details the dissertation objectives subjacent to the described problems that emerge from the described scenario.

1.3. DISSERTATION OBJECTIVES

The dissertation main objectives are:

- The request, done by clients, that specify what data is needed from the database, should contain *'what information is needed'* and the less possible about '*how*' the information is obtained. 'How' related aspects like database location and technology should be transparent to the clients.

- *Changes that occur in the database* should not necessarily be propagated to all clients. In this way, clients are not aware of the database changes, either syntactic (ex.: change of a table name) or structural (added or deleted table).

- The local databases vocabulary should be hidden, providing a *common vocabulary* across several databases. Developers should be aware of the information model and unaware of complex database schemas.

- The developed solution should allow the development of applications that improves and allows integration with other applications, decoupling the GUI interface from the database. This is a particular integration solution between the GUI and the database. Therefore, application functions that usually manipulate/access the data that is stored in the database should be able to be reused by other heterogeneous applications.

The above mentioned four objectives should be a step further in the development of loosely coupled applications, reducing maintenance when changes occur in databases and also improving developer's productivity by abstracting from specific database details.

1.4. THE APPROACH IN A NUTSHELL

In order to achieve the above-mentioned objectives, this dissertation explores the development of middleware using two emerging technologies: *Semantic Web Technologies* (SWT) (Buchmann et al., 2006) and *Service Oriented Architectures* (SOA) (W3C-SW, 2006) through Web Services (WS) (W3C-WS, 2006).

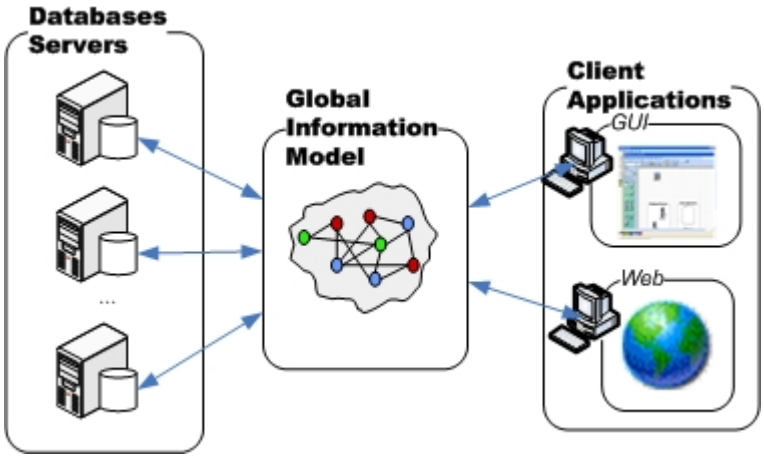


Figure 2 - Integration using a shared information model

As illustrated in Figure 2, the middleware (middle-tier located between the "front-end" tier and the database-tier) contains the global virtual view over a set of databases. At the left part of Figure 2 is illustrated the entire set of databases existent in the organization. At the right part of the Figure 2 is illustrated the set of "front-end" tiers which in this case are Web based and GUI (applications that are "windows" like). In this case, one "front-end" application can be connected to one or more databases.

The global information model (information model, or global virtual view, which are considered synonyms), illustrated in the centre of Figure 2, represents the virtual view over the entire set of database schemas, describing what data is accessible.

This approach is going to be described in detail in chapter 3.

1.5. PUBLICATIONS

Two articles related with this dissertation research have been produced.

One article was accepted and published in the "International Journal of Interoperability in Business Information Systems" (IBIS) journal, issue 3. The other article was presented and published in the "International Conference on Enterprise Information Systems" (ICEIS) 2007.

Regarding to the article published in IBIS journal, it is entitled "Using Semantic Web Technologies to Build Adaptable Enterprise Information Systems" (pages 41-46 of issue 3) and can be found at (Caires and Cardoso, 2006). According to IBIS site, the IBIS journal aims at distributing latest research results of the interoperability domain free of charge. Hence, all accepted issues of the journal can be downloaded for free at this website. All accepted articles are peer reviewed by at least three independent members of the review board. In order to assure an excellent quality, the journal only accepts highly rated articles. Accepted submissions are published in the journal and citable. The journal has got an international ISSN number allowing a unique identification of papers.

Regarding to the article published and presented in the ICEIS 2007, it is entitled "Using Semantic Web and Service Oriented Technologies to Build Loosely Coupled Systems: Swoat – A Service and Semantic Web Oriented Architecture Technology" and can be found at (Caires and Cardoso, 2007). According to ICEIS site, ICEIS is well known for its prestigious keynote speakers; we are proud to have hosted more than 60 distinguished keynote speakers in the previous 9th editions of this conference. The conference has a double-blind review process and after a strict selection procedure all accepted papers will be published in the proceedings (paperback and CD-ROM) under an ISBN, and indexed by ISI Proceedings, INSPEC and DBLP.

1.6. DISSERTATION ORGANIZATION

This dissertation is organized in six chapters.

Chapter I is the dissertation introduction.

Chapter II describes the background technologies and theories used in this dissertation. Integration, Semantic Web Technologies (SWT) and Service Oriented Architectures (SOA) will be described. It will also describe the main advantages and characteristics of each technology in the context of its application in the developed framework.

Chapter III describes the developed framework named Swoat (Service and Semantic Web Oriented ArchiTecture). Therefore, the Swoat architecture, the way clients invoke services, the request and response messages will be described in this chapter.

Chapter IV demonstrates a running example based on Swoat. Thus, it is described the deploy environment in which Swoat is used. This chapter details a "front-end" application the uses the developed framework Swoat.

Chapter V states the state-of-the-art of integration products that allow a global view over a set of databases allowing integration. It splits the products in categories and explains the characteristics of each product.

Chapter VI describes the conclusions and future work.

2. BACKGROUND

> "Words - so innocent and powerless as they are, as standing in a dictionary, how potent for good and evil they become in the hands of one who knows how to combine them."
> —Nathaniel Hawthorne (1804-1864)

This chapter presents a description of the theories and concepts that represent the foundation of the proposed solution, named Swoat framework. As already presented in the previous chapter, Swoat framework main objectives are:

- Clients requests containing what data is needed should focus on 'what' related issues and not on 'how' related issues.
- Avoid the propagation of database syntactic and structural changes to "front-end" clients.
- Hide the database vocabulary.
- Provide and allow reusability of the implemented functions by heterogeneous systems.

Swoat is intended to be deployed between the database system and the "front-end" tier. Consequently, the connection between the database and the "front-end" tier can be seen as integration among two systems: database system (ex. database in MySql) and the "front-end" tier system (ex. developed in Java). Thus, this chapter starts by

exposing several existing integration concepts and types. From all the integration concepts presented, Enterprise Application Integration (EII) is presented in detail because it is used in the Swoat framework. One important component used to achieve EII integration is middleware, which is deployed between databases and "front-end" clients. This way, clients connect with the databases mediated by the middleware that provides a centralized access point to several databases.

This chapter follows with another technology used on the prototype: Semantic Web (SW). One key concept of SW is ontology, which provides a way to formally specify the information model (information model and domain model are considered synonyms in this dissertation) of the organization. This model is particularly important because it describes the business domain of the organizations, also providing a virtual view over the entire set of database tables that are used to store the data described by the information model (domain model). Concerning with the languages that allow specifying an information model, this chapter presents several formal languages. After the formal specification of the model, one important issue is the possibility to query it. Consequently, ontology query languages (equivalent to SQL query languages in databases) are presented.

One interesting issue when integrating databases with the "front-end" through the use of middleware is related with the way the "front-end" tier interacts with the middleware that stores the information model, providing a global view over the entire set of databases. This means that the middleware which provides a global and virtual view over the entire set of systems should be able to provide services to other "front-end" tiers that can be invoked even by heterogeneous systems. This way, the provision of services by the middleware, following a Service Oriented Architecture (SOA) approach is the solution that is adopted in the Swoat framework. Both SOA and Web Services technologies (Web Service is used to implement a SOA system) will be outlined in this chapter.

2.1. INTEGRATION

The integration definition that is considered in this dissertation is "the task of making disparate applications work together to produce a unified set of functionality" (Hohpe and Woolf, 2004). Since applications (software systems), likely run on multiple computers, which may represent multiple platforms, and may be geographically distributed, integration is needed in order to allow connection among systems. Some of the applications may be run outside the enterprise by business partners or customers, which difficult the integration process. Also, applications might not have been designed with integration in mind and are difficult to change.

There exist several ways to solve an integration problem (Alexiev et al., 2005). The first solution is to build ad-hoc import/export bridges that interface between two applications. In most cases, these bridges are one-way, which means that data is transported from one application to the other, but not necessarily the other way around. These bridges are typically incorporated into either the exporting or importing application.

A more structural solution approach is to develop a general mapping (of concepts) between two applications. Mapping is necessary in integration because the existing heterogeneity between applications. These (one-way or two-way) mappings have a major disadvantage: the probable high number point-to-point mappings that need to be maintained.

Another option is to base the mappings on a central repository. The central repository defines all concepts and their interrelations in an independent way. The main advantage of such a solution is that the repository can be developed and maintained as a central store of information on the top of individual applications.

These approaches have been used as the base for several integration concepts defined by the following keywords: Enterprise Application Integration (EAI), Enterprise Information Integration (EII) and Data Integration (DI).

DI is essentially extraction, transformation and loading (ETL) of data from disparate systems into a single data store for the purpose of manipulation and reporting (Taylor, 2006). DI is related with import/export. In practice, DI occurs, for example,

when a database table is replicated from one database (used by a specific application) to another application database. As an example, if the employees table is used in both systems, it can be replicated from the Human Resource system to the Account system.

EII creates virtual data integration between various data sources. EII is useful when a business or organization needs to create a common gateway with one access point and one access language to disparate data sources. It is related with the creation of a central repository of concepts, generally called the global virtual view. Using EII integration, all the sources remain in their original databases and applications access the global virtual view. Since this integration type is used in the Swoat framework, it is going to be described in detail more ahead in this section.

EAI is a business computing term for the plans, methods and tools aimed at modernizing, consolidating and coordinating the computer applications of an enterprise (Ruggiero, 2005). One feature of EAI is its ability to track and deliver changes to the proper application or system. EAI is useful when connecting two or more applications in real time and is related with "mappings between applications". For example, it is not enough to replicate database tables from one database to another. It is also necessary to synchronize them. Thus, EAI announces a set techniques and also patterns (Hohpe and Woolf, 2004) in order to achieve this. In the literature, EAI is seen as a high level integration concept, therefore considering both DI and EII (described in the next paragraph) as a subset of EAI.

The main differences between DI, EAI and EII are:

- DI is essentially related with importing/exporting data across several databases.

- Using an EII approach, the data remains in the sources and it is the global and virtual view that describes the entire set of database data that is stored, thought an information model (global virtual view).

- EAI is a high level concept (some references consider DI and EII as a subset of EAI) that comprises the integration of application essentially by mapping database concepts between applications in order to keep them synchronized.

Independently of the integration concepts used to solve an integration problem, integration can be categorized in several types. From a high level perspective, each integration concepts (DI, EAI, and EII) can implement one of more integration type, presented ahead.

2.1.1.Integration Types

Multiple approaches for integrating applications have evolved over time, which can be summed up in six types of integration (Hohpe and Woolf, 2004): Information Portals; Data Replication; Shared business functions; Service-oriented architectures; Distributed business processes; Business-to-business integration.

These integration types represent high-level integration approaches that can be implemented in the above mentioned integration solutions (EAI, EII and DI).

2.1.1.1.Information Portals

The Enterprise Information Portal (EIP), also known as a business portal, is a concept for a Web site that serves as a single gateway to a company's information and knowledge base for employees and possibly for customers, business partners, and the general public as well.

Because many business users have to access more than one system to answer specific question or to perform a single business function, information portals aggregate information from multiple sources into a single display to avoid having the user access multiple systems for information. Actually, in the *.com* age, it is frequent to face this integration need.

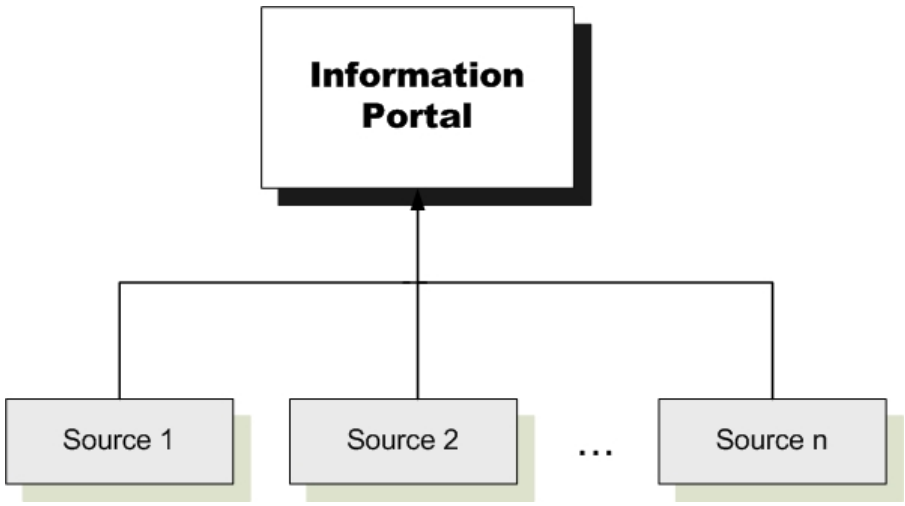


Figure 3 - Information portal

Figure 3 illustrates an example of the information portal integration type. The information portal accesses to data stored in several sources. More information can be found in (Firestone, 2002).

2.1.1.2.Data Replication

Data replication is faced when many business systems require access to the same data. For example, a customer's address may be used in the customer care system, in the accounting system and the billing system. When customers address changes, it has to be changed in all systems (updating the copy of the address database table).

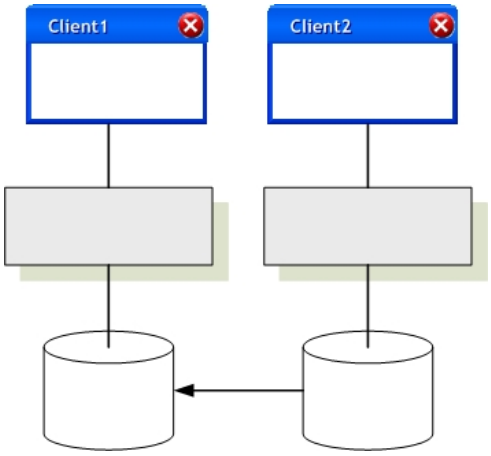


Figure 4 - Data replication

Figure 4 depicts two applications that are architecturally organized in a GUI (top of the figure), the "function provider" (middle of the figure) and the database (bottom of the figure). Since there exists common data between the two systems, the data is transferred from one system to another as illustrated by the arrowed connector between the two databases. More information about this topic can be found at (Buretta, 1997).

2.1.1.3.Shared Business Function

In the same way that many business applications store redundant data, they also tend to implement redundant functionality. When multiple systems need to check if the address matches the specified postal code it makes sense to expose the functions as a shared business function that is implemented once and available as a service on other systems.

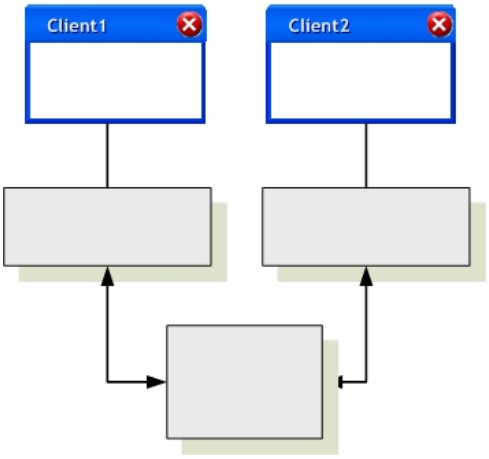


Figure 5 - Shared data function

Figure 5 illustrates two applications that access to a shared business functions. In this case it is the function provider that has the responsibility to access to the specified shared functions (bottom of the figure).

2.1.1.4.Service Oriented Architecture

Shared business functions are often referred as services. A service is a well-defined function that is available and responds to requests from service consumers. A new application can be developed using existing remote services that may be provided by other applications. Therefore, calling a service can be considered integration between the two applications.

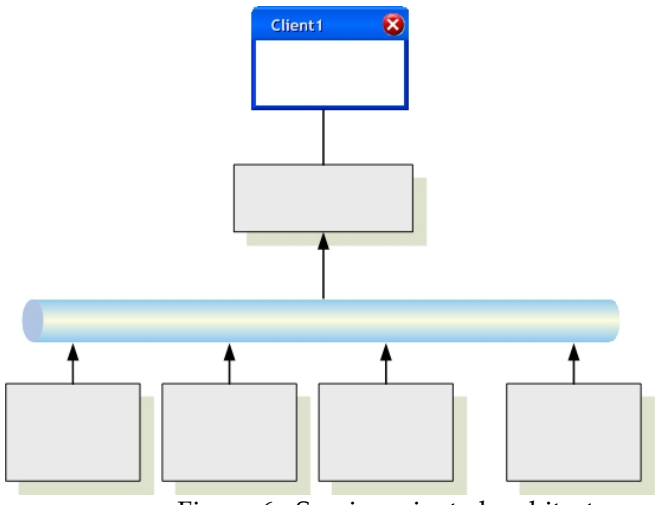


Figure 6 - Service oriented architecture

Figure 6 illustrates an application (top of figure) that access to several services. The application business provider is itself a service that in this case is accessible to other applications (illustrated in the bottom part of figure). More information about SOA can be found at (Erl, 2005).

2.1.1.5.Distributed Business

A simple business function such as "placing an order" can easily touch half a dozen systems. In most cases, all relevant functions are incorporated inside existing applications. What is missing is the coordination between these applications. Therefore, it can be added a business process management component that manages the execution of a business function across multiple existing systems. The boundaries between an SOA and a distributed business can be fuzzy. For example, all relevant business functions can be exposed as services and then encode the business process inside the application that accesses all services via an SOA.

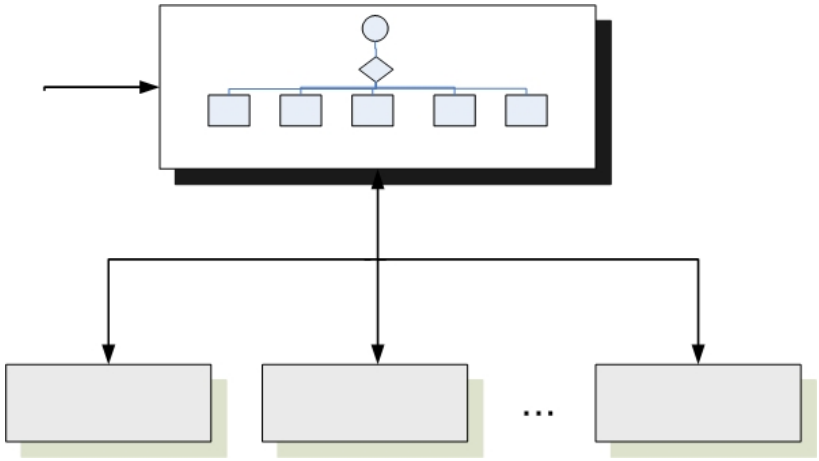


Figure 7 - Distributed business process

The business process manager (illustrated in the top of the Figure 7) manages the execution of a business function across multiple existing systems.

2.1.1.6.Business-to-Business Integration

In a broad sense, Business-to-Business (B2B) integration refers to all business activities of an enterprise that have to do with electronic messages exchange between it and one or more of its trading partners (Bussler, 2003). In addition, B2B integration encompasses direct peer-to-peer exchange of messages between enterprises (without intermediary) as well as their interaction in marketplaces as sellers or buyers (where the marketplace is the intermediary). B2B also refers to the software technology that is the infrastructure to connect any back-end application system within enterprises to all its trading partners over formal message exchange protocols like Electronic Data Interchange (EDI) or RosettaNet (RosettaNet).

In practice, B2B integration is very useful because in many cases, business functions may be available from outside suppliers or business partners. For example, the

shipping company may provide a service for customers to compute shipping cost or track shipments. This is an example of a business to business integration.

Figure 8 - Business to business integration

Figure 8 illustrates the case in which a business function is accessed and made accessible to and from an external organization (each square represents an organization). This is the more general and ambitious case of an integration solution.

More information about B2B can be found at (Bussler, 2003).

The following section presents a basic building block of the developed solution (Swoat): EII.

2.1.2.Enterprise Information Integration

This section presents Enterprise Information Integration (EII) that is the integration approach implemented in the Swoat framework. EII is the integration of data and application functionality from multiple systems into a unified, consistent and accurate representation geared toward the viewing and manipulation of data. Data is aggregated, restructured and relabelled (if necessary) and presented to the user. Information integration is targeted at end users who are required to deal with multiple systems in order to perform their given tasks (Taylor, 2004). Typical examples of these needs are the "front end" developers of applications.

Providing a unified view of data from disparate systems comes with a unique set of requirements and constraints. First, the data should be accessible in "real-time" – meaning that it should be accessing the systems directly as opposed to accessing stale data from a previously captured snapshot. Second, the semantics, or meaning, of data needs to be resolved across systems. Different systems may represent the data with different labels and formats that are relevant to their respective uses, but that require some sort of correlation by the end user in order to be useful to them.

The unified view represents the information model of an organization (or a specific domain), containing relationships and rules that represent the semantics of the data and it's interaction with other data and processes. Meaningful information is achieved by describing relationships between business entities (that store data in data

entities – database tables) and the rules that govern its use. This is described as "business rules" and has previously been captured in the programming code of a system. With a unified view, most of the relationships and rules can be captured directly in an information model.

One of the most important aspects in the creation of the unified view is mappings (the specification of the correspondence between the data at the sources and those in the global schema that represents the unified view). Such a correspondence is modelled through the notion of mapping. It is exactly this correspondence that will determine how the queries posed to the system are answered.

Two basic approaches for specifying the mapping have been proposed in the literature, named Local-as-View (LAV) and Global-as-View (GAV) (Lanzerini, 2002). GAV approach requires that the global schema is expressed in terms of the data sources, as depicted in Figure 9. On the other side, the LAV requires the global schema to be specified independently from the sources (Figure 10), and the relationships between the global schema and the sources are established by defining every source as a view over the global schema.

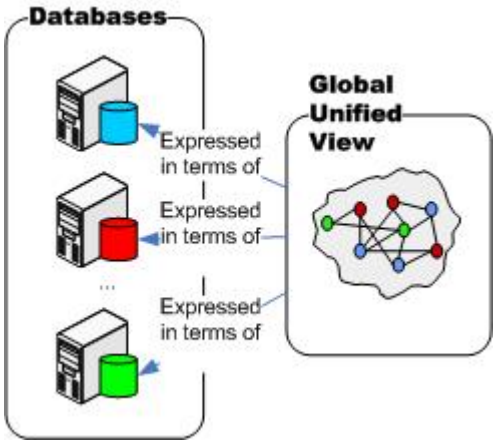


Figure 9 – GAV approach - The global schema is expressed in terms of the data sources

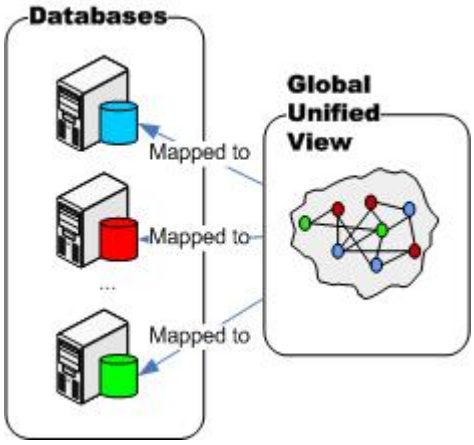


Figure 10 – LAV approach - The global schema is specified independently from the sources

It is well known that processing queries in the LAV approach is a difficult task. Indeed, in this approach the only knowledge we have about the data, in the global schema, is through the views representing the sources, and such views provide only partial information about the data. On the other hand, query processing looks easier in the GAV approach, where we can take advantage that the mapping directly

specifies which source queries correspond to the elements of the global schema (Lanzerini, 2002).

Another difference between the two approaches is that while in the LAV approach the designer may concentrate on declaratively specifying the content of the sources in terms of the global schema, in GAV approach, one is forced to specify how to get the data from the global schema by means of queries over the sources. The term "unified view" is used as a synonym for global virtual view (GVV). The GVV represents the information model or domain model (these two concepts are also considered synonyms).

The following section presents an important technology used in Swoat that is used to specify the information model of the organization: Semantic Web technologies through one stack component named ontologies.

2.2. SEMANTIC WEB TECHNOLOGIES

This section describes Semantic Web Technologies (SWT) stack. Regarding to the Swoat framework (the developed prototype), one component of the stack used in Swoat is ontologies, that allow the formal specification of the information model. Ontologies are suitable to provide a possible solution to describe the information model of the organization, describing in the context of Swoat framework, what data is accessible.

Since it is fundamental to query the information model in order to extract data from it, ontology query languages are also explained (an ontology query language is like SQL to relational databases) and presented as a way to do so. Thus, Simple Protocol and RDF Query Language (SPARQL-W3C) query language is detailed because it is used in Swoat framework.

Considering that Web Ontology Language (OWL) (W3C-SW, 2006) is the chose language used in Swoat framework to specify the information model, this language is going to be detailed more ahead in this chapter.

Referring to Semantic Web technologies, in order to explain the roots that originated Semantic Web technologies (SWT), currently, the World Wide Web (WWW) is primarily composed of documents written in Hyper Text Markup Language (HTML), a language that is useful for publishing information. During the first decade of its existence, most of the information on the Web is designed for human consumption. Humans can read Web pages and understand them, but for inherent meaning it is not shown in a way that allows their interpretation by computers (Cardoso and Sheth, 2006b). The information on the Web can be defined in a way that it can be used by computers not only for display purposes, but also for interoperability and integration between systems and applications. One way to enable machine-to-machine exchange and automated processing is to provide the information in such a way that computers can understand it. This is precisely the objective of the semantic Web.

According to the inventor of World Wide Web also related to the Semantic Web definition, the semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation (Berners-Lee et al., 2001). It is based on the idea of having data on the

Web defined and linked such that it can be used for more effective discovery, automation, integration, and reuse across various applications. The Semantic Web provides an infrastructure that enables not just Web pages, but databases, services, programs, personal devices, and even household appliances to both consume and produce data on the Web (Hendler et al., 2002).

The following sections described each important component in the SW stack.

2.2.1.The Semantic Web Stack

The semantic Web stack is composed of a series of standards organized into a certain structure that represents their interrelationships. This architecture was first proposed by Tim Berners-Lee.

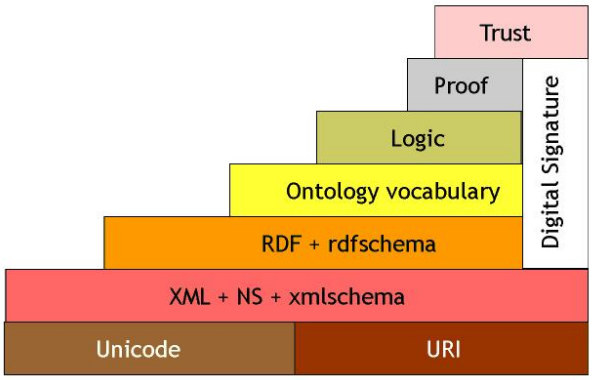


Figure 11 - The semantic Web stack (Berners-Lee, 2000)

Figure 11 illustrates the different parts of the semantic Web architecture. The base of the stack is the Unicode and the Universal Resource Identifier (URI). On the top is the XML layer, which in turn underlies RDF and RDF Schema (RDFS). Web ontology languages are built on top of RDF and RDFS. The three last layers are logic, proof and trust, which have not been significantly explored. Some of the layers rely on the digital signature component to ensure security, as illustrated in the vertical layer.

In the remaining of this section is explained each technology of the semantic Web stack.

2.2.1.1.Universal Resource Identifier and Unicode

A Universal Resource Identifier (URI) is a formatted string that serves as a mean of identifying abstract or physical resource. A URI can be further classified as a Uniform

Resource Locator (URL) or a Uniform Resource Name (URN). A URL identifies resources via a representation of their primary access mechanism and a URN remains globally unique and persistent even when the resource ceases to exist or becomes unavailable.

Unicode provides a unique number for every character, independently of the underlying platform or program. Before the creation of Unicode, there were various different encoding systems making the manipulation of data complex and required computers to support many different encodings.

2.2.1.2.XML, Namespaces and Xml Schema

XML is accepted as a standard for data interchanging, allowing the structuring of data but without communicating the meaning of the data. It is a language for semi-structured data and has been proposed as a solution for data integration problems, because it allows a flexible coding and display of data, by using metadata to describe the structure of data.

Namespaces are defined by the W3C Namespaces in XML Recommendation as a collection of XML elements and attributes often referred to as an XML "vocabulary". One of the primary motivations for defining an XML namespace is to avoid naming conflicts when using and re-using multiple vocabularies.

The structure of data is represented through *XML Schema* (W3Schools, 2006). XML Schema is a language for restricting the structure of XML documents. One of the greatest strength of XML Schemas is the support for data types, which turns easier:

- To describe allowable document content.
- To validate the correctness of data.
- To work with data from a database.
- To define data facets (restrictions on data).
- To define data patterns (data formats).
- To convert data between different data types.

In conclusion, an XML Schema defines:

- Elements that can appear in a document.

- Attributes that can appear in a document.
- Which elements are child elements.
- The order of child elements.
- The number of child elements.
- Whether an element is empty or can include text.
- Data types for elements and attributes.
- Default and fixed values for elements and attributes.

2.2.1.3.RDF and RDF Schema

RDF (Resource Description Framework) is essentially a meta-model in which its basic building block is an object-attribute-value triple, called a statement (Antoniou and Harmelen, 2004). While objects are resources, and describe for example, authors and books, properties allow describing relations between objects, for example, "*written by*" and "age". The value presents the content of the property. For example, 'Book' *hasAuthor* 'José' means that the object 'Book' has an author whose name is José.

RDF is a universal language that lets users describe objects using their own vocabularies. RDF does not make assumptions about any particular application domain, nor does it define the semantics of any domain. It is up to RDFS to do so.

Referring to RDF Schema, it provides a type system for RDF. It is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes and allowing users to define resources with classes, properties and values (W3C-Schema, 2004). The concept of class in RDFS is similar to the concept of class in java. A class is a structure of similar things and inheritance is allowed. An RDFS property (rdfs:Property) can be viewed as an attribute of a class.

Summing-up, RDF and RDF schema (RDFS) main characteristics are:

- RDF provides a foundation for representing and processing metadata.
- RDF has a graph-based data model. Its key concepts are resource, property and statement. A statement is a resource-property-value triple.

- RDF has a XML-based syntax to support syntactic interoperability. XML and RDF complement each other because RDF supports semantic interoperability.
- RDF has a decentralized philosophy and allows incremental building of knowledge, and its sharing and reuse.
- RDF is domain-independent. RDF schema provides a mechanism for describing specific domains.
- RDF schema is a primitive ontology language. It offers certain modelling primitives with fixed meaning. Key concepts of RDF schema are class, subclass relations, property, sub property relations, and domain and range restrictions.

More information can be found at (RDF, 2007).

The following section details OWL, the formal language used in Swoat.

2.2.1.4.Ontology

Toward the main objective of the semantic Web (well defined meaning of information), a fundamental keyword in the centre of the semantic Web is ontology.

Ontology is a "specification of a conceptualization" (Gruber, 1993). Explaining each keyword of the definition separately, first, a conceptualization is the way we think about a specific domain and second, a specification provides a formal way of writing it down. Third, formally writing it down allows that it can be understood both by humans and computers. Ontologies intend mainly to explicit formal specifications of the terms in the domain and relations among them, therefore defining a common vocabulary for researchers who need to share information in a domain.

The following figure, Figure 12, illustrated one example of ontology that relates the person concept with the address concept (both domain classes) through the property *hasAddress*. The Person concept contains two attributes: name (the name of the person) and birthDate (the birth date of the person) and the address concept also contains two attributes: address (the description of the address) and the type of the address (ex. residential, holidays, etc)

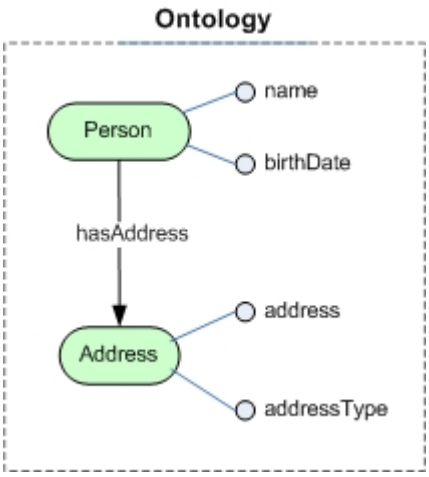


Figure 12: Sample ontology

Some of the reasons to develop a ontology are (Noy and McGuinness, 2001):

- *To share common understanding of the structure of information among people or software agents*: This is one of the more common goals in developing ontologies (Musen 1992; Gruber 1993). Ontology defines the common language making possible that computer agents can extract and aggregate information.

- *To enable reuse of domain knowledge*: was one of the driving forces behind recent surge in ontology research. There also exists the possibility of combining several already defined ontologies.

- *To make domain assumptions explicit:* underlying an implementation makes it possible to change these assumptions easily if our knowledge about the domain changes. In addition, explicit specifications of domain knowledge are useful for new users who must learn what terms in the domain mean.

- *To analyze domain knowledge:* is possible once a declarative specification of the terms is available. Formal analysis of terms is extremely valuable when both attempting to reuse existing ontologies and extending them (McGuinness et al. 2000).

Many disciplines now develop standardized ontologies that domain experts can use to share and annotate information in their fields. Medicine, for example, has produced large, standardized, structured vocabularies such as SNOMED (Price and Spackman 2000) and the semantic network of the Unified Medical Language System (Humphreys and Lindberg 1993). Broad general-purpose ontologies are emerging as well. For example, the United Nations Development Program and Dun & Bradstreet

combined their efforts to develop the UNSPSC ontology which provides terminology for products and services.

2.2.1.5.Logic, Prof and Trust

The Logic, Prof and Trust layers are on the top of the ontology layer. *Logic* layer purpose is to provide similar features to the ones that can be found in the First Order Logic (FOL). The idea is to state any logical principle and to allow the computer to reason by inference using these principles. *Proof* traces or explains the steps involved in logical reasoning. *Trust* in the top layer of the Semantic Web architecture provides authentication of identity and evidence of the trustworthiness of data and services. While the other layers of the semantic Web stack have received a fair amount of attention, no significant research has been carried out in the context of this layer (Cardoso and Sheth, 2006a).

The following section describes in formal languages to build ontologies and details OWL.

2.2.2.Formal Languages to Describe an Ontology

Several formal languages can be used to express ontologies. For instance CycL (CyCorp, 2006), KIF (Genesereth, 2006) and RDF (Lassila and Swick, 1999) and OWL (W3C-SW, 2006), that is going to be described in detail later on this section, are some examples.

2.2.2.1.CycL

CycL was developed by Cycorp and it's a formal language whose syntax derives from first-order predicate calculus and from Lisp. CycL is used to express common sense knowledge and to represent the knowledge stored in the CycL Knowledge Base. CycL's major strengths are expressiveness, precision, meaning and use-neutral representation and its major weakness is the focus within the Cyc project has been on the engineering of large common sense knowledge bases, and not on the advancement of deductive reasoning technology.

More information can be found at (Cycl, 2007).

2.2.2.2.KIF

It was originally created by Michael Genesereth and others participating in the DARPA Knowledge Sharing Project. Knowledge Interchange Format (KIF) is a

language designed for use in the interchange of knowledge among disparate computer systems (created by different programmers, at different times, in different languages, and so forth). KIF was created to serve as syntax for first-order logic that is easy for computers to process. KIF features full semantic expressiveness. One inconvenience of this language is its high computational complexity. Although the original KIF group intended to submit to a formal standards body, that did not occur.

More information can be found at (KIF, 2007).

2.2.2.3.RDF(S)

Please see the previous section RDF and RDFS Schema to more information about this topic.

2.2.2.4.OWL

Until the development of OWL, existing languages to express ontologies have demonstrated some confinements in its expressiveness to formally specify complex ontologies. Therefore, a number of research groups in both United States and Europe had already identified the need for a more powerful ontology modelling language. This led to a join initiative to define a richer language, called DAML+OIL - the name is a join of the names of the U.S. proposal DAML-ONT and the European language OIL (DAML, 2001). DAML+OIL in turn was taken as the starting point for the W3C Web Ontology Working Group in defining OWL, the language that is aimed to be standardized and broadly accepted ontology language of the Semantic Web. OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web, adding more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties and characteristics of properties (e.g. symmetry), and enumerated classes.

The OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans (McGuinness and Harmelen, 2004).

OWL ontology consists of Individuals, Properties and Classes. *Individuals* represent objects in the domain that we are interested in. It is also know as the Universe Of Discourse (UoD). Individuals are also known as instances. *Properties* are binary relations on individuals. They are also known as roles in description logic and relations in UML. *Classes* are interpreted as sets that contain individuals. They are

described using formal descriptions that state precisely the requirements for membership of the class.

OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users: OWL Lite, OWL DL and OWL Full (Antoniou and Harmelen, 2004).

OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. OWL Lite also has a lower formal complexity than OWL DL.

OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class).

OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Figure 13 depicts the relation between expressiveness and inference capabilities from the ontology. Inference refers to the abstract process of deriving additional information from the ontology. The higher the expressiveness the less is guaranteed that the ontology has computational and inference guarantees.

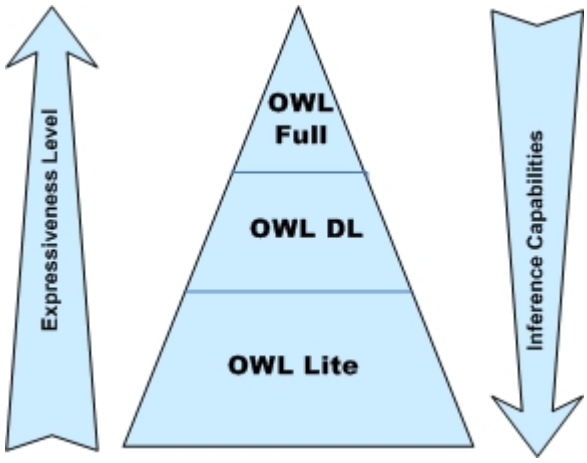


Figure 13 - OWL species

The following sections described existent ontology query languages and details SPARQL.

2.2.3.Ontology Query Languages

Several languages to query ontologies exist, namely: RQL, RDQL, N3, SeRQL, SPARQL, Versa, Triple, SquishQL, RxPath and RDFQL (Haase et al., 2004). The work on query languages has been progressing for a number of years. Several different approaches have been tried, ranging from familiar looking SQL-style syntaxes, such as RDQL and Squish to specific ones. Of all approaches, those that emulate SQL syntactically have probably been the most popular and widely implemented.

The latest ontology query language is SPARQL (SPARQL Query Language for RDF, W3C), which is an effort of standardization to OWL query languages, by W3C. A standardized query language offers developers and end users a way to write and to consume the results of queries across this wide range of information (SPARQL-W3C, 2005).

In total, SPARQL consists of a query language, a means of conveying a query to a query processor service, and the XML format in which query results will be returned. There are a number of issues that SPARQL does not address yet, most notably, SPARQL is read-only and cannot modify a dataset (Dodds, 2005).

More information about SPARQL can be found at (SPARQL-W3C, 2005).

2.3. SERVICE ORIENTED ARCHITECTURE AND WEB SERVICES

When developing software it is extremely useful to implement solutions that allow interoperability with heterogeneous solutions. This means that independently from the location, from the implementation language and from the technological platform, systems should allow integration with other solutions. Systems can take advantage of Service Oriented Architectures (SOA) which means that a system can provide services that can be invoked by other applications. In order to achieve this issue, Web Services (WS) are actually the most prominent technology to implement a SOA system. Since SOA and WS are used in the developed prototype (Swoat), this section describes these two technologies.

To cope with the restrictions of more traditional distributed objects architectures (ex. DCOM and Java RMI), in the early 2000's the concept of service-oriented architectures (SOA) was introduced. SOA describes an approach which facilitates the development and composition of modular services that can be easily integrated. According to W3C, a SOA is a set of components which can be invoked, and whose interface descriptions can be published and discovered (Pennington et al., 2007).

Most distributed computing technologies have the concept of services and are defined by interfaces. There are many different possibilities for developing SOA. Web services (WS), Java RMI, DCOM and CORBA are some examples. However, Web service is the preferred solution because it eliminates many of the interoperability problems between applications and services (Pennington et al., 2007).

The following sections describe SOA and WS.

2.3.1. Service Oriented Architectures

Service Oriented Architecture (SOA) is an architectural style for building software applications that use and share services available in a network such as the Web. Following SOA architecture, applications provide services to other systems. Defining the service keyword, it is an implementation of well-defined business functionality, and such services can then be consumed by clients in different applications or business processes. As illustrated in Figure 14, services are invoked by sending a request to the service provider. As the consequence from the service invocation, the service response will contain the result.

Referring to SOA main advantages, it allows reusing existing assets where new services can be created from an existing IT infrastructure of systems. In other words, it enables businesses to leverage existing investments by allowing them to reuse existing applications, and promises interoperability between heterogeneous applications and technologies.

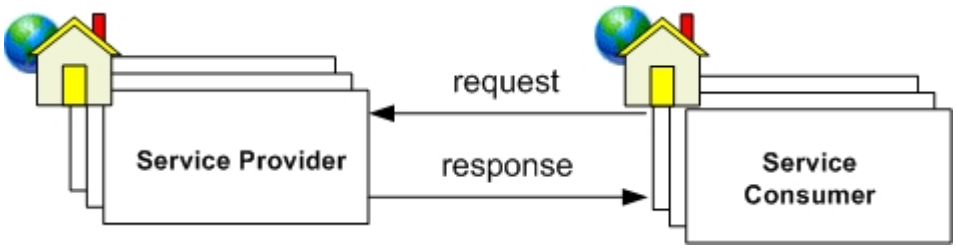


Figure 14 - Service provider and consumer

SOA provides a level of flexibility that wasn't possible before in the sense that (Mahmoud, 2005):

- Services are software components with well-defined interfaces that are implementation-independent. An important aspect of SOA is the separation of the service interface (the what) from its implementation (the how). Such services are consumed by clients that are not concerned with how these services will execute their requests.
- Services are self-contained (perform predetermined tasks) and loosely coupled (for independence).
- Services can be dynamically discovered.
- Composite services can be built from aggregates of other services.

Describing each of the above mentioned topics, *modular* means that components can be reused and it is possible to compose them into larger components. *Available* means services must be exposed outside of the particular paradigm or system in which they are available in. Also, services can be completely decentralized and distributed over the Internet, having a machine-readable description that can be used to identify the interface of the service. Additionally, the *service interface* is independent of the implementation and describes what services are available and how to invoke them. Another service characteristic is that they are made available in a repository where users can find the service and use the description to access the service.

Regarding to SOA advantages in software development and the main differences from existent solutions, SOA differs from existing distributed technologies in that most vendors accept it and have an application or platform suite that enables SOA. This way, it enables changes to applications while keeping clients or service consumers isolated from evolutionary changes that happen in the service implementation. Finally, SOA provides enterprises better flexibility in building applications and business processes in an agile manner by leveraging existing application infrastructure to compose new services (Kodali, 2005).

The next section describes Web Services as an important technology for implementing a SOA based system.

2.3.2.Web Services

A Web service is designed to support interoperable machine-to-machine interaction over a network (W3C-WS, 2006). Web Services are self-contained, self-describing, modular applications that can be published, located and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes.

In fact, it is sometimes useful to consider the benefits of these standards as two separate aspects. For simplicity, two aspects of Web Services will be considered: the Web aspect and the Service aspect (Fremantle et al., 2002).

Web aspects:

- Web-based protocols: Web services based on SOAP-over-HTTP are designed to work over the public Internet. The use of HTTP for transport means these

protocols can transverse firewalls, and can work in a heterogeneous environment.

- Interoperability: SOAP defines a common standard that allows differing systems to interoperate.

- XML-based: The XML is a standard framework for creating machine-readable documents. Managed by the W3C, XML is an open Web standard for creating interoperable standard documents.

Services aspects (Fremantle et al., 2002):

- Modular: Service components are useful in themselves, reusable, and it is possible to compose them into larger components.

- Available: Services are available to systems that wish to use them. Services must be exposed outside of the particular paradigm or system they are available in.

- Described: Services have a machine-readable description that can be used to identify the interface (that is, what sort of service it is), and its location and access information (that is, where it is).

- Implementation-Independent: The service interface must be available that is independent of the ultimate implementation. For example, SOAP messages can be hosted by almost any technology.

- Published: Service descriptions are made available in a repository where users can find the service and use the description to access the service.

The following section illustrates the Web Service Stack, focusing on SOAP, WSDL and UDDI.

2.3.2.1.Web Services Stack

Web Services stack is composed by three main layers: Network, Messaging and Description, as illustrated in Figure 15.

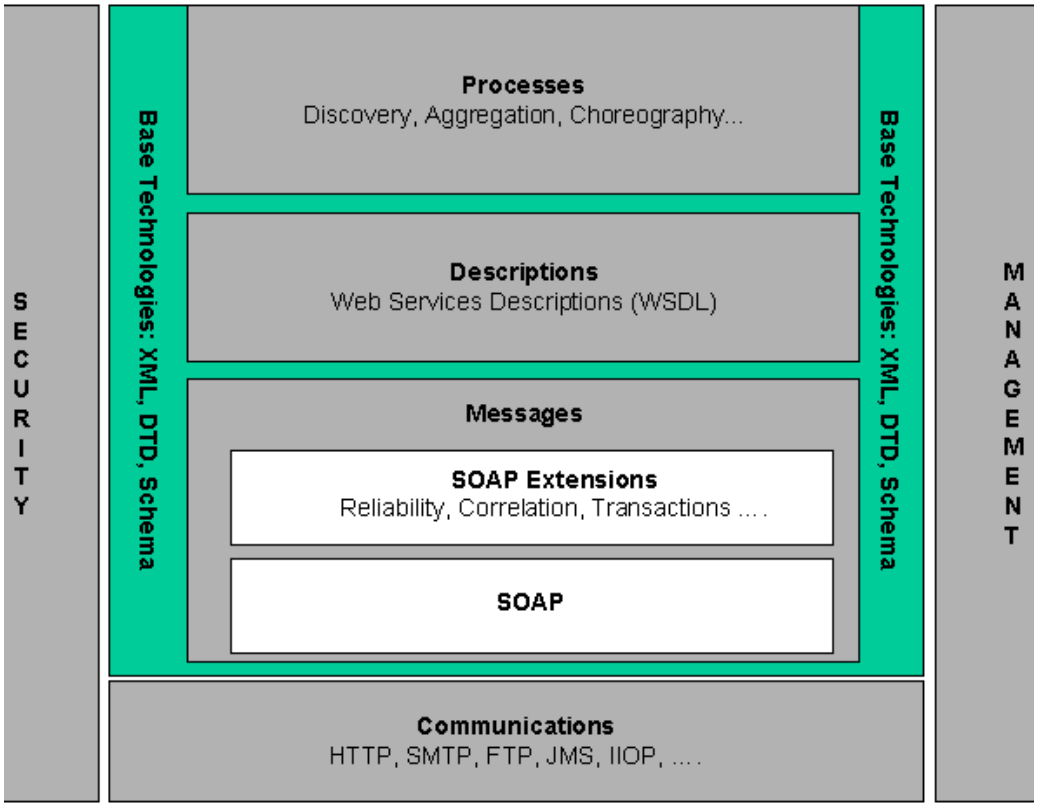


Figure 15 - Web services stack (WSA-W3C, February 2004)

The next sections illustrate the key languages to implement the Messages, Descriptions and Processes (focussing on the Discovery) layers.

2.3.2.2.Simple Object Access Protocol

The Simple Object Access Protocol (SOAP) is typically understood to be a request-response mechanism, based on HTTP. SOAP starts out as just XML message format: an envelope, which contains an address, some headers and a body. The body consists of one or more elements. The elements may be encoded using a standard SOAP encoding, more simply stated, a standard way of capturing programming language data elements – integers, doubles, strings – in a common interoperable format. Also, SOAP can be sent over a transport- typically HTTP (W3C-SOAP, 2003).

SOAP is the key to interoperability, because almost every vendor, both large and small, supports it.

2.3.2.3.Web Service Description Language

WSDL is an XML format for describing network services (Christensen et al., 2001). As communications protocols and message formats are standardized in the Web community, it becomes increasingly possible and important to be able to describe the

communications in some structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. The Web Services Description Language offers the ability to describe the inputs and outputs of a Web Service. It allows a server to publish the interface of a service, so a client knows that if it sends a SOAP message in format A to the service, it will receive a reply in format B.

2.3.2.4.Discovery: Universal Description, Discovery and Integration

UDDI is a platform-independent, XML-based registry for businesses worldwide to publish services on the Internet. UDDI is an open industry initiative, sponsored by OASIS, enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet (Wikipedia). UDDI it both a standard and a set of public implementations hosted by companies such as IBM and Microsoft. These public UDDI implementations can be used by any Internet-connected business to publish, search for, and locate business and services.

2.3.2.5.Security and Management

Threats to Web services involve the host system, the application and the entire network infrastructure. To secure Web services, a range of XML-based security mechanisms are needed to solve problems related to authentication, role-based access control, distributed security policy enforcement, message layer security that accommodate the presence of intermediaries. At this time, there are no broadly-adopted specifications for Web services security. As a result developers can either build up services that do not use these capabilities or can develop ad-hoc solutions that may lead to interoperability problems. Web services implementations may require point-to-point and/or end-to-end security mechanisms, depending upon the degree of threat or risk. Traditional, connection-oriented, point-to-point security mechanisms may not meet the end-to-end security requirements of Web services. However, security is a balance of assessed risk and cost of countermeasures. Depending on implementers risk tolerance, point-to-point transport level security can provide enough security countermeasures (WSA-W3C, February 2004).

Web service management is the management of Web services through a set of management capabilities that enable monitoring, controlling, and reporting of service qualities and service usage. Such service qualities include health qualities such as availability (presence and number of service instances) and performance (e.g. access

latency and failure rates), and also accessibility (of endpoints). Facets of service usage information that may be managed include frequency, duration, scope, functional extent, and access authorization. A Web service becomes manageable when it exposes a set of management operations that support management capabilities. These management capabilities realize their monitoring, controlling and reporting functions with the assistance of a management information model that models various types of service usage and service quality information associated with management of the Web service. Typical information types include request and response counts, begin and end timers, lifecycle states, entity identifiers (e.g. of senders, receivers, contexts, messages, etc.).

2.4. CONCLUSION

From a narrower perspective, the connection between a database and a "front-end" application can be seen as *integration*. Thus, integration concepts and theories were presented in order to understand how to take advantage of them when applied to this particular scenario. Regarding to the integration theory, three main concepts exist: EAI, DI and EII. EAI is a high level integration concept, therefore describing plans, terms and tools to solve integration between two or more applications. DI refers mainly to distributing (importing/exporting) data to several databases. EII refers to the creating of a global and virtual view over a set of systems, allowing integration. The following integration types were also presented: Data replication; Shared business function; Service Oriented Architecture; Distributed business; Business to business integration. Swoat framework was implemented in order to provide three types of integration, service-oriented-architecture (sharing application functions); distributed business and business-to-business integration (allowing integration with other organizations).

Integrating a database with a "front-end" can take advantage of the newest integration concept: EII. EII states that one "front-end" application can be connected to one or more databases through a global and virtual view (other synonyms are domain model and information model) over the entire set of databases. The global and virtual view contains essentially concepts (like person, employee, etc) and its relations among concepts and it is stored in the middleware that is deployed between the databases and the "front-end" tier.

The Semantic Web theory provides one concept that can be used in the definition of this information model: ontology. In order to formally specify ontology several languages exist. CycL, KIF, RDF(S) and OWL are possibilities to do so. Since OWL is a W3C recommendation and adopted in Swoat, it was described in more detail.

After implementing the information model one fundamental feature is the ability to query and extract information from it. This is when query languages come in. SPARQL (that is to OWL, what SQL is to relational databases) was described as the W3C standardization effort for a query language.

The provision of a global view over a set of databases should be accessible by heterogeneous "font-end" applications (GUI and Web based). Accordingly to SOA (Service Oriented Architecture) advantages, it is a suitable solution in promoting loosely coupling between software components so that they can be reused. Several advantages were described when adopting SOA architecture. SOA architecture can be implemented using Web Services. Why are Web Services important in software development? Several advantages were stated, highlighting the open standards and the advantages inherited from Web and from Services.

3. SWOAT FRAMEWORK

> "Great things are not done by impulse, but by a series of small things brought together."
> — Vincent van Gogh (1853-1890)

This chapter details Swoat (Service and Semantic Web Oriented ArchiTecture) framework. Therefore, it is organized in three main sections:

- *Requirements*: This section describes Swoat framework requirements, intended to be deployed between "front-end" and databases tiers.
- *Architecture/Design*: This section describes Swoat architectural and design issues, presenting its layered architecture and the technologies used in each layer. A global view of Swoat architecture is also presented in this chapter, combining all the technologies used in Swoat framework.
- *Implementation*: This section presents Swoat implementation technologies, focussing on Swoat three layered architecture and highlighting each layer implementation details. It also presents the mappings from the information model to the database, and proceeds with Swoat services requests. It ends with the presentation of the methodology that should be followed in order to deploy Swoat.

The following section describes Swoat requirements.

3.1. REQUIREMENTS

Several types of requirements can be defined for a software system. Examples are functional and non-functional, user requirements and system requirements (Sommerville, 2001). This section will describe the high level system requirements in natural language.

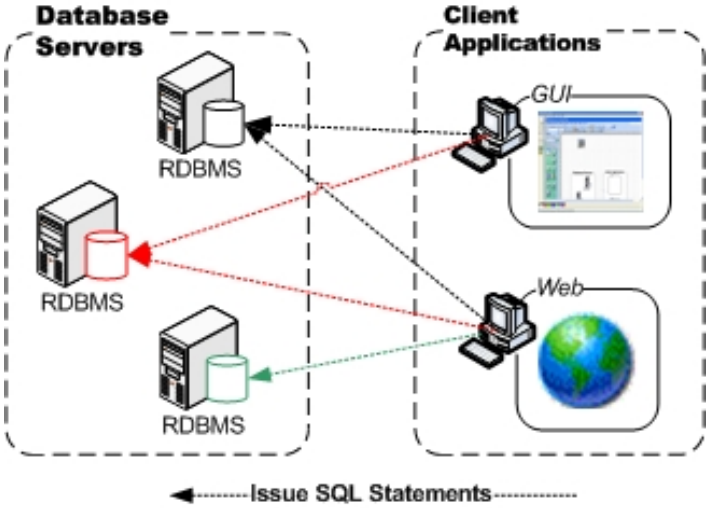


Figure 16 – "Front-end" tier directly connected to databases

Swoat intends to be a solution to the following main requirements:

- The request done by clients ("front-end" tier) (illustrated in the right part of Figure 16), formulated to get the required data (stored at the left part of Figure 17), should specify *'what information is needed'* and the less possible about 'how' the information is obtained. 'How' related aspects like database location and technology should be transparent to the clients.

- *Changes that occur in the database* should not necessarily be propagated to all clients. In this way, clients are not aware of the database changes, either syntactic (ex.: change of a table name) or structural (added or deleted table).

- The local databases vocabulary should be hidden, providing a *common vocabulary* across several databases. Developers should be aware of the information model and unaware of complex database schemas.

- The solution should allow the development of applications that improves and allows integration with other applications, decoupling the GUI interface from

the database. Therefore, application functions, usually implemented to manipulate/access the data that is stored in the database should be able to be reused by other heterogeneous applications.

The following section describes the proposed architecture in order to achieve the above mentioned requirements.

3.2. ARCHITECTURE/DESIGN

This section starts by explaining the affinity between the concepts and technologies that picture the theoretical foundation of the solution presented: Swoat - a Service and Semantic Web Oriented ArchiTecture. From a high level perspective, as illustrated in Figure 17, Swoat represents an implementation of the *middle-tier* (illustrated in the centre of the figure) that is deployed between the *database tier* (illustrated in left part of the figure) and the *client tier* (illustrated in right part of the figure). The database tier contains the databases that store the data and the "front-end" tier represents client applications that invoke the services available by Swoat.

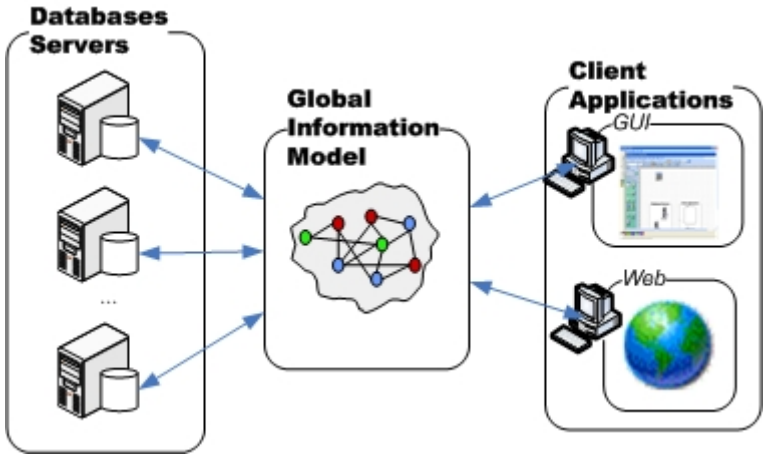


Figure 17 – Three tier architecture

This section presents the advantages that motivated *Enterprise Information Integration* (see section 2.1.2) adoption in Swoat. In the chosen integration solution, one common aspect that should not be disdained is the way systems are connected (integrated) to databases. In this field, *middleware* is presented as the technology used to mediate the connections between systems. It is also a building block of the EII presented solution.

One important issue when implementing EII solutions is *Service Oriented Architecture* (SOA) – allowing the SOA integration type and business-to-business type integration. The main advantages of using SOA in the Swoat context will be described. Through middleware, one interesting Swoat feature is the provision of services to "front-end" tier, allowing that heterogeneous clients invoke Swoat services. The middleware centralizes the information model (the represents the domain of the problem that the application intends to solve) describes what data can be accessed. *Semantic Web*

Technologies (SWT) – mainly by using one stack component that is named ontology - will be presented as a solution to describe, store and formalize the organization information model. The advantages of using SWT and particularly *ontologies* are outlined.

After the presentation of Swoat theoretical foundation, Swoat layered architecture is illustrated devoting on its three layers.

3.2.1.Swoat Technologies and Approach

This section describes the technologies used in Swoat, as illustrated in Figure 18. Each technology and its advantages are described in the next sections.

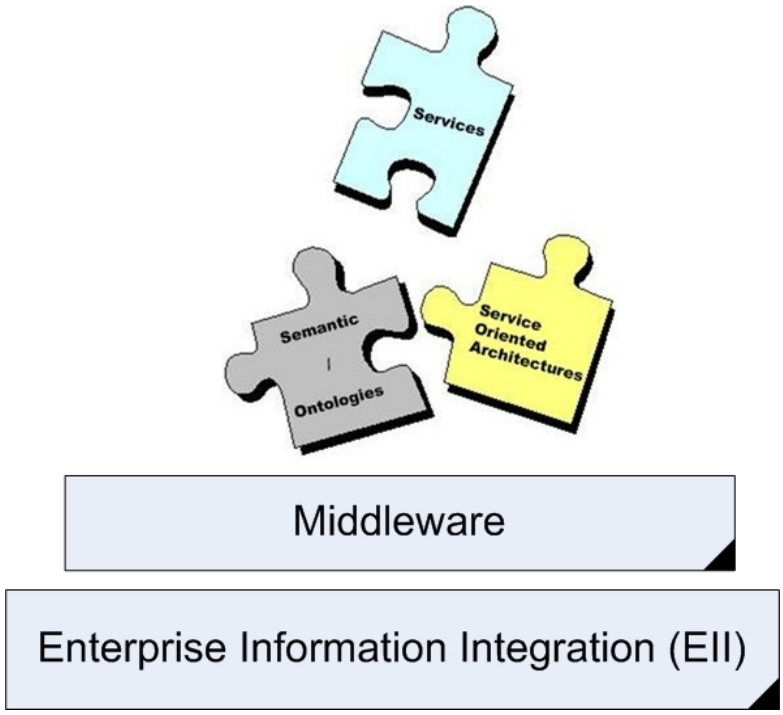


Figure 18 - Swoat implementation technologies

One characteristic of *EII* (the Swoat integration approach), depicted at the bottom of Figure 18, is the fact that the global information model is centralized and accessible by all software systems, based on open standards. It is the global information model that provides the description of the data stored on the database, abstracting the technical user from issues like for example, where is located the database and the name of the table.

Based on the EII integration approach and by using middleware to implement an EII integration solution, two key technologies are used in order to implement EII integration: Service Oriented Architecture, Services and Semantic. All the presented

technologies and concepts are combined in order to achieve the requirements presented in the previous section. Briefly referring to the technologies and its main purpose that motivated its choice:

- *EII*: This integration approach is used because the database sources remain on their original databases and it is provided a global and integrated view over the entire database set. This way, there is not need to implement data import/export and synchronization mechanisms between databases.

- *Middleware*: "front-end" tiers do not connected directly to one or more databases but instead connect to the global and virtual view stored in the middleware. This means that changes that occur in the databases (which are connected to one or more clients) are not necessarily propagated to all clients connected.

- *Service Oriented Architecture and Services*: The middleware should provide services to "front-end" tier independently of its implementation language. This means that any data required by the client should be obtained by service invocation and not by issuing SQL queries in databases. This allows that any application invoke services, allowing this way integration with other applications.

- *Semantic/Ontologies*: The global and virtual view is implemented using W3C formal languages to specify ontologies. This means the global view is accessible to clients, describing all the data that can be accessible.

The following section describes each technology used and refers to the main reasons that motivated its choice. It starts by the integration approach used to integrate, in this particular context, databases to "front-end" tiers.

3.2.1.1.Integration Approach

Swoat relies on EII integration approach. The EII approach states that the data resides in its original sources, and is accessible by created a virtual view over the entire set of databases. This virtual view not only describes what data is accessible but also provides an integrated view over the entire set of databases. Concerned with the creation of a global view, one important question is related with the approach to choose. Two approaches exists namely GAV or a LAV and when choosing the integration approach one important question should be answered: what is the best

approach in EII integration referring to Swoat context? In order to answer the question, the GAV and LAV are going to be briefly explained.

Whereas information model in the GAV approach is limited to the "as-is" database schema in each database, in the LAV approach the programmer/annalist/manager has the possibility of create the "to be" model. For example, organizations may often need to change organization processes that may reflect in also changing databases, either syntactically or structurally, in order to adapt them to the new reality. In this case, the information model is quickly changed in the GAV approach while in the other case, in the LAV, it is slower due the fact that it has to be changed manually. However, one important question should not be underestimated: the clients connected. In the GAV approach the programmer has no control over the generated model because it is built based upon the existing database schemas. This can be seen has a disadvantage in some scenarios.

Regarding to Swoat framework, what was the choice: GAV or LAV? Due to the characteristics of the problem to address, the LAV approach was the solution. The reasons that motivated the choice were:

- The virtual view, referred as the information model, stays intact when changes occur in the databases.
- Adding or deleting data source means adding or deleting mappings from the information model concepts (that need to be added to the global view) to the database source tables.
- The global view is implemented independently from the local sources. This means that the concepts used in the global model may not be the same as the ones implemented in the systems/databases by programmers.
- Even if changes occur in the local sources (internal or COTS) the global view does not need to be changed (this is a characteristic of a GAV approach).
- If changes occur in the global virtual view, databases may not have to be changed due.

Combining EII with the LAV approach, the following advantages came forth:

- The centralized information model represents the logical model designed by knowledge workers and not by programmers. It represents the global view that top managers intend for the organization.

- Most of the business logic can be described in the centralized information model and not on the lines of code of the application.

- It is focused on the end users productivity. The centralized information model represents the information that is stored in the databases, abstracting from technical and application specific semantics.

The main disadvantage of the adopted solution is that it EII is a relatively new theoretical concept and consequently it has not yet been demonstrated that existing technologies are capable of implementing an EII solution in a sustainable way.

The following section describes middleware as a solution to implement EII, providing a global and centralized view to systems that can be internal or external "front-end" tiers.

3.2.1.2.Middleware

Middleware (or middle-tier, these are synonyms in this dissertation) is used in order to accomplish the information model centralization, allowing integration of both data and application functionality. In the Swoat context, middleware is deployed between the database tier and the client tier ("front-end"), as illustrated in Figure 19. As also illustrated in figure, the middleware is organized in three layers (that are going to be detailed in the next section named "Swoat Architecture – Global View"): database layer (responsible for connecting with the databases), domain layer (stores the information model) and service layer (used by clients to invoke services).

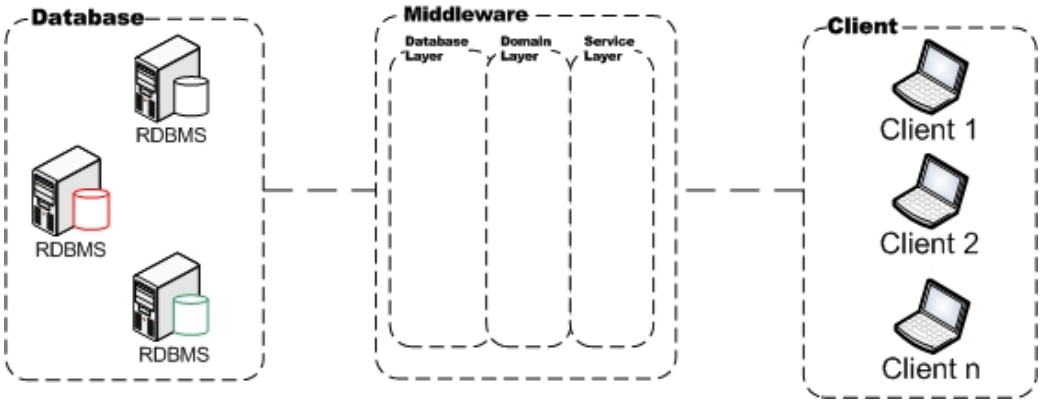


Figure 19 - Example of middleware between databases and clients

Some of the advantages that emerge when using middleware between applications and databases are:

- When changes occur in the database, those changes are not propagated to all clients (that can be several). This occurs because the database and the clients are not directed connected but instead mediated by the middleware.

- Middleware helps organizing and centralizing the business logic avoiding reimplementation across systems that connect to the same database.

- The substitution of the database technology without having to change all clients. It is the middleware are is responsible for interacting with the database. If the database technology changes, it is expected that only the middleware has to be changes, and not all the connected applications.

However middleware can also lead to some disadvantages:

- The development time of an application increases.

- Using a middleware to access the database is slower that direct database access.

- The middleware may not avoid the propagation of changes that occur in database, increasing in some situations the maintenance time.

In Swoat context, how is middleware used in EII solution? The information model (sometimes referred as global view) is stored in the middleware that is accessible by all "front-end" tiers. It is the middleware that centralizes the global view, providing an integrated view to all data sources. When describing middleware solutions, these are mainly characterized by the type of interaction/connection that is made by clients that interact with the middleware. This way, two types of middleware exist: synchronous or asynchronous. While in the first type the client waits for the result of the request, in the second, the request is stored in a queue and the client continues its execution. The asynchronous type is not suitable for all applications because the client may need the result in order to continue its execution. Mainly due to this issue, the solution presented in this dissertation uses the synchronous type.

The following section describes the Service Oriented architecture that is followed in Swoat framework. Since clients interact with the middleware through service invocations (implementing a SOA), the next section also refers to services.

3.2.1.3.Service Oriented Architecture and Services

Using SOA (Service Oriented Architecture), clients can interact with the middleware through service invocation, as illustrated in Figure 20. Examples of services that clients can invoke are: "change the address of a person" or "cancel an order". While the first service simply returns data, the second one is much more complex, representing several activities. Due to the different types of services that can exist, Swoat service types are described more ahead in this chapter.

SOA was chosen to be used in the proposed solution due the advantages that it brings in achieving structuring, loose coupling and standardization of business functionality among interacting software applications. Applications invoke a series of discrete services in order to perform a certain task (Pennington et al., 2007).

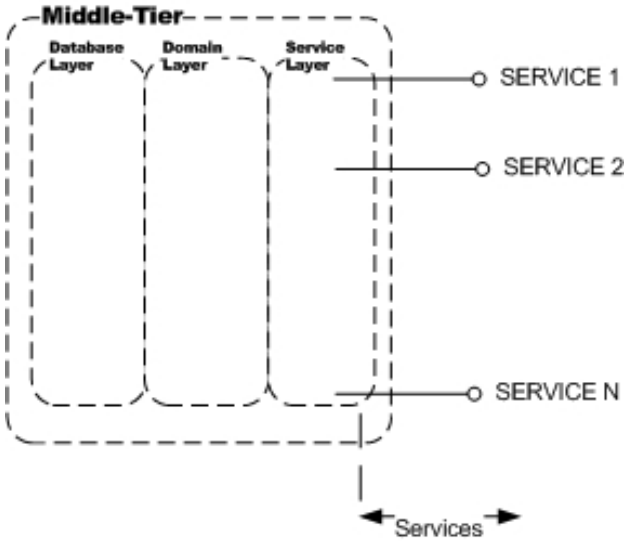


Figure 20 - Services provided by the middleware

Being a SOA solution, the middleware should provide services to clients. Service provision is responsibility of the presentation layer, as illustrated in Figure 20. For more information about SOA and services see section 2.3.

The next section presents Semantic Web technologies and focuses especially in ontologies that are used in Swoat framework in order to describe and implement the information model.

3.2.1.4.Semantic Web Technologies

Ontologies, one component of Semantic Web technologies (as described in the background chapter), allow the formal description of an information model. Swoat

recurs to ontologies to describe the information model of the organization. For example, ontology can be used to describe the "Personal Data" information model. This model describes the information that could be requested to the middleware through service invocation. Using a formal model, "front-end" tiers can read the formal model, locate what information they need and build the message in order to invoke the service. The result of the service invocation represents the data that was asked. The model can be shared or reused by other organizations that need to interact – it contains just the concepts and the relation among them. Also, several already defined ontologies exist to describe particular domain, which can be reused.

For more information about SWT please see section 2.2.

The following section describes a global view of all the presented technologies used in Swoat framework.

3.2.2.Swoat Architecture – Global View

Swoat is essentially a middleware that allows EII integration in which the main purpose is to be deployed between databases and "front-end" clients. In an EII solution, SOA allows "front-end" clients, either internal or external from the organization, to invoke services. This way, application data and functionality is accessed by service invocation. Therefore, the proposed solution is a SOA middleware, being accessible through service invocation.

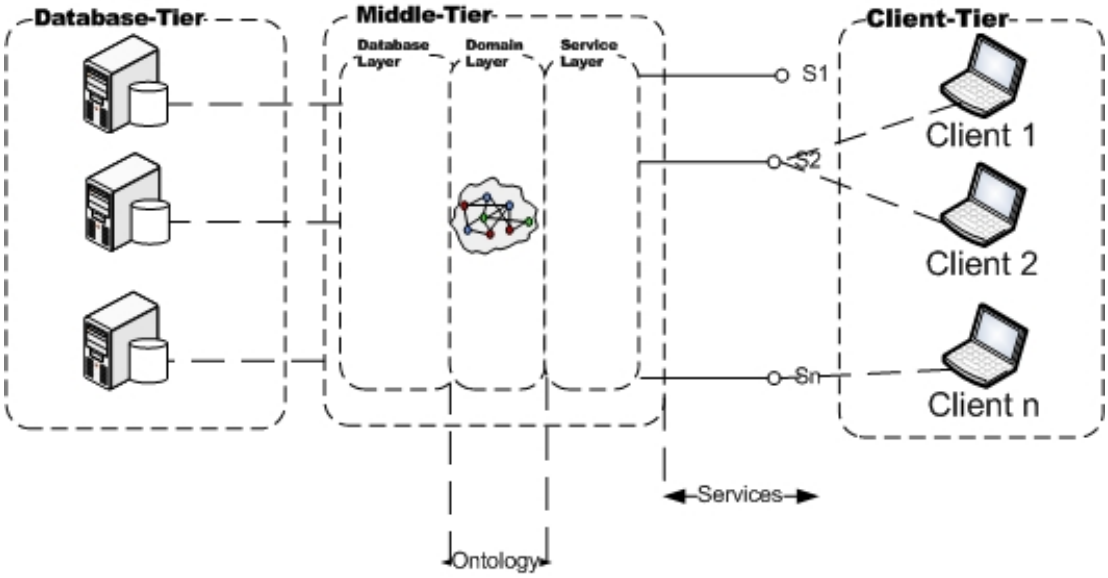


Figure 21 – Swoat high level architecture

One key issue of the middleware is to provide a global and virtual view over the entire set of databases. The global view, stored in the middleware, describes the

organization data stored in the databases and provides a mean of communication with technical users that need data. This means that technical users (programmers) request the needed data by invoking middleware services. In the case of simple data services (just return data from the database), the data can be accessed is described by the information model.

In the context of Semantic Web Technologies (SWT), information model is referred as ontology. The ontology describes the information model and has the advantage of formalizing it in a specific language. This turns the model accessible for both humans and computers.

Storing and centralizing the domain mode, and being accessible through service invocation, Swoat, illustrated in Figure 21, is composed of three tiers: Database (left part of the figure), Middleware (or middle-tier – centre of the figure) and Clients tiers (right part of the figure). Due to the complexity that the middleware tier can acquire, one way of organization it is through the use of layering. Layering is one of the most common techniques that software designers use to break apart a complicated software system (it is something like "divide to conquer").

Typical layer systems are based on three main layers:

Data source

This layer is responsible for allowing access to database systems. The following section, related with implementation issues, presents the technology used to achieve database access.

Domain

This layer stores the information model (that described classes, attributes and relation between classes – also referred as ontology) implemented using a formal language. The formal model should not contain any technical details but instead be an accurate representation of the information model of the organization (usually closely related with the business area of the organization, plus the support sections like human resource, account, etc). This way, it should be possible to reuse and or distribute the implemented model. Usually the information model is not dissociated from the databases, which store the data described. This means that the concepts formally described in the information model should be mapped to the database table that stores the corresponding data. This way, one important issue in Swoat context is to map the information model to the databases. Therefore, the information model and the mappings, described above, should be stored in this layer:

- Domain classes to database tables: one concept in the information model should be mapped to al least one database table. For example, the domain class *Person* should be mapped to the database table name *customer*.

- Domain class attributes to database classes attributes: one attribute in a domain class should be mapped to at least one table attribute. For example, the domain class attribute, named *name* (stores the name of the person) should be mapped to the database attribute named *description* stored in the customer table.

- Domain relation to database relations: The existing relations in the information model should be mapped to existing relations in the database (relation between tables – using keys). For example, if the domain class person is related with the class address through the relation property named hasAddress, this property should be mapped to the fields that relate the two corresponding tables (customer and address).

The mappings should contain the necessary information in order to build the SQL statement to get the required data, thought service invocation. The types of services are described in the next section.

Presentation

This layer is responsible for proving services to clients ("front-end" tiers). Three types of services are available through the presentation layer.

The first type is "*Data Services*". Data services allow "front-end" tiers to access to data stored in the databases by invoking services. Example if this type of service is the getGender service, that returns all gender types stored in the database.

The second type of service is called "*Functional Service*" that allow to expose application functions through services. What distinguishes data services from function services is the fact that function services entail business rules validation. Several definitions about business rules exist but in this context the definition is "rules that must be verified in order to save and retrieve data from the database". An example of a functional service is byProduct. This service reduces the stock of the product but should only allow the stock reduction if there exist sufficient quantity available. So, there is a data service invocation but also some processing (stock calculation). Another example is the getPersonalInformation service that returns the person information (name, birth date, address and contact). If the person hasn't all the

information in the database then the service should also return a message indicating what is missing (for example, the address).

The third type of service is the "*Business Services*". It is the combination of several data and functional services. These three types of services are illustrated in Figure 22.

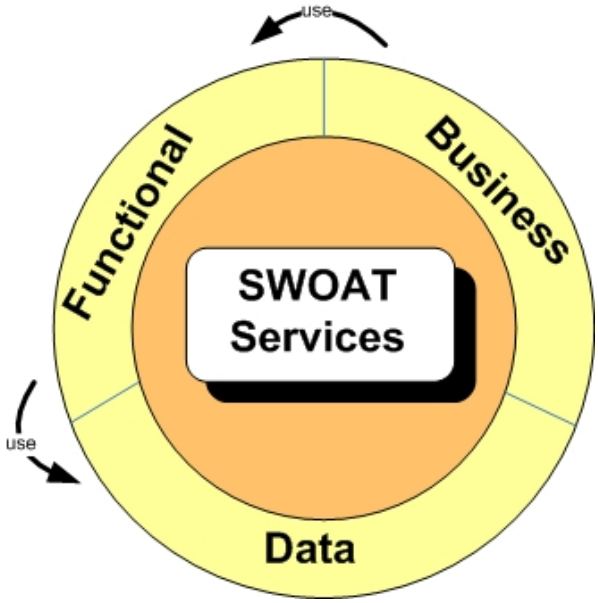


Figure 22 - Presentation layer (types of services)

Referring to the data service, these are the basic services and recurrently used by the other two types of services in order to request data. Due to the high number of data services that can exist (on for each table and the combination between them), Swoat provides a service named getGenericService() that simply returns the specified data. This service allows users to specify the request using a neutral language, independently from the database technology and from the formal language used to specify the model. This means that if changes occur in these technologies, it should not be necessary to change all clients that invoke Swoat services. Additionally, the language used by clients (that should be the request in order to get the data), named "Neutral Client Query Language" (NCQL) should allow users to specify:

- Fields (domain attributes) to be returned. For example, return the gender description and gender abbreviation.
- The order of the output fields (ascending- ASC, descending- DESC). Example, return the gender sorted ascendant.
- Filters (for example, return only names started by letter A). For example, only the gender description that have abbreviation.

The NCQL is going to be detailed in the implementation section, more ahead in this chapter.

The following tables crosses the requirements with the technologies used to achieve it. The symbol √ illustrates that the technology is used to satisfy the requirement.

Technologies / Requirements	**EII**	**Middleware**	**Service Oriented Architecture**	**Services**	**Semantic / Ontologies**
Clients should only specify 'what' information is needed and the less possible about 'how' related issues				√	
Avoid the propagation of database changes to clients	√	√			
Hide local database vocabulary from clients					√
Allow to reuse application function by other application			√	√	

Table 3.1: Requirements and its relation with technologies.

3.3. IMPLEMENTATION

This section details Swoat implementation issues. Therefore, it starts by describing the Swoat architecture, presenting each layer. It also presents the methodology that should be followed when deploying Swoat in software development. Mappings from the information model to the database tables are also presented, finishing with issues related with Swoat service invocation by clients.

3.3.1.Swoat Framework

Swoat middleware, which is a layered solution, is organized in three layers: Data Source Layer (DSL), Business Layer (BL) and Presentation Layer (or Service Layer that are considered synonyms) (PL). As illustrated in Figure 23, DSL is identified by (1), BL by (4) and DSL by (11). Each layer is going to be described separately, referring to the numbers illustrated in Figure 23 to facilitate the presentation.

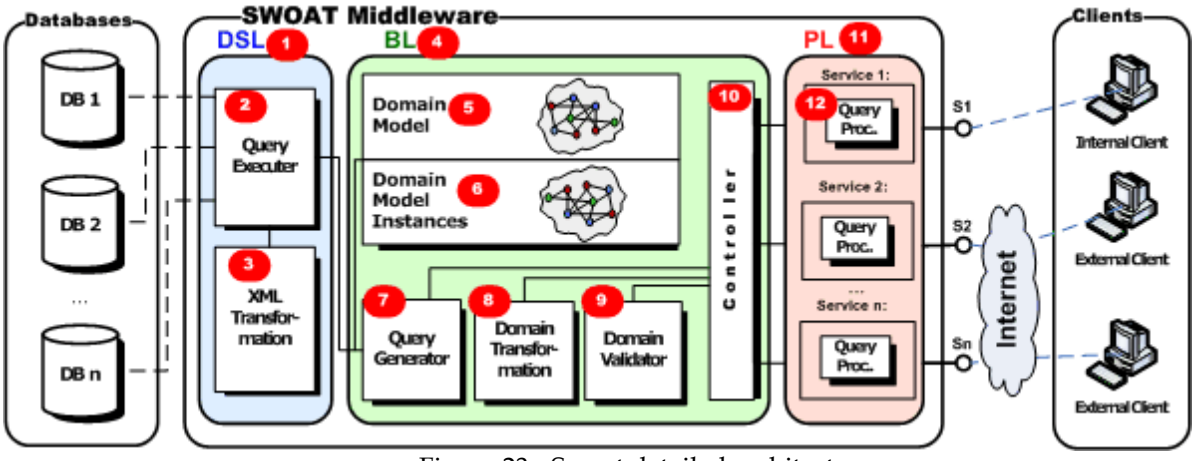


Figure 23 - Swoat detailed architecture

3.3.1.1.Data source Layer

This layer (1) is responsible for the communication with the relational database management systems. In Swoat, this layer is implemented using Hibernate (Hibernate, 2006), which is an open-source product developed in java. It increases the developer productivity enabling developers to focus more on the business problem (Hibernate, 2006). It is interoperable with any JDBC compliant database and supports

more than 20 popular dialects of SQL including Oracle, DB2, Sybase, MS SQL Server, PostgreSQL, MySQL, HypersonicSQL, Mckoi SQL, SAP DB, Interbase, Pointbase, Progress, FrontBase, Ingres, Informix and Firebird (Hibernate, 2006).

The Query Executer (2) allows executing SQL queries on the databases. The returned data from the 'Query Executer' (2) is then transformed to XML in the 'XML Transformation' (3). Two main reasons motivated this transformation. The first one is that having the data structured in XML is easier to manipulate it in the 'Business Layer' (4), with the final objective of returning XML to the clients. The other reason is that with XML we decouple the DSL from the BL, which means that independently from the implementation of the DSL, the only thing that has to be assured is the XML structure. This means that the Hibernate technology can be changes with having to change the business layer.

3.3.1.2.Business Layer

The BL (4) is where the information model, formally specified in OWL (language used to specify an ontology) is stored. The domain model (information model), (5) represents the important concepts (ex. Person, Address) its attributes (ex. age), and relations between concepts (ex. one person has one address). The business entities (or concepts) are mapped to the DSL (that allows access to the database) by creating instances (6) of the OWL model. The instances contain the necessary information (database, table and attribute) in order to build SQL queries that allow retrieving the required data from the databases. It is the 'Query Generator' (7) that is responsible for extracting the necessary information from the 'Domain Model Instances' (6) and 'Domain Model' (5) and generate the SQL statement that is going to be executed in the 'Query Executer' (2) of the DSL. The language used to query the OWL, in order to extract the data to build the SQL expression, is SPARQL.

The 'Controller' (10) is responsible for interacting with the 'Query Generator' (7) in order to obtain the returned data, formatted in XML. The data returned from the DSL is then transformed, in the 'Domain Transformation' (8), and structured in a format that reflects the OWL model. For example, if is specified in the domain model (information model) that one person has at least one address, the result XML will reflect this hierarchical structure.

The 'Domain Validator' (9) is responsible for validating the data accordingly to the specified business rules (ex. one person must have at least one address, which means that a person record without the address is not a valid record). In this case, the

business rules represent the validations that have to be verified in order to insert and retrieve data from the databases.

The main component in the business layer is the ontology. Consequently, when building the ontology that resides in the domain layer, important domain classes and respective attributes are be described. For example, the class Person can contain the name and the birth date attributes. The class Address can contain the address and the address type attributes. Also, the relation between the Person and Address should be specified, as illustrated in Figure 24.

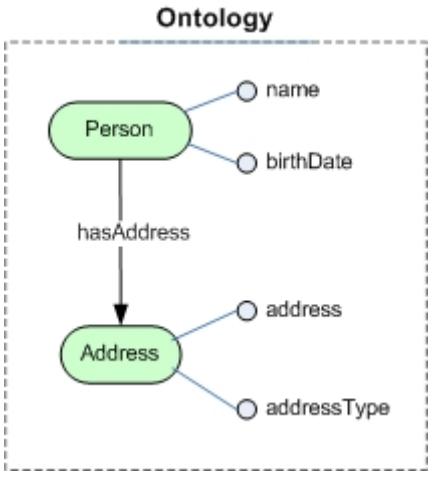


Figure 24 - Person and Address concepts related by the hasAddress property

Ontology relations describe the relations that exist between domain concepts. For example, the domain class Person is connected to Address through the property hasAddress. The ontology also contains rules. For example, we can use the ontology to describe the relation between the address and the Person classes using a cardinality restriction: one person must have at least one address. Swoat properties should always start by has{propertyName} on in the case of the inverse, is{propertyName}Of. Examples are hasAddress of isAddressOf.

Classes, attributes and relations between classes are implemented using OWL (a W3C standard) language.

3.3.1.3.Presentation Layer

The PL (11) provides services to "front-end" tiers (client applications). Therefore, this layer is responsible for receiving and processing the service requests from the clients. The presentation layer also returns and manipulates information that is described in

the domain model (information model). Requests are structured in XML and contain terms that are described in the domain model, as is going to be described in detail in the next section.

The 'Query Processor' (12) is responsible for validating the request from the client, and interacting with the 'Controller' (10) in order to get the desired data. Requests and responses are encapsulated in SOAP, and exposed as Web Services.

In Swoat, data services are allowed through the invocation of a Swoat service named getGenericService. The user request message contains all the information that is required by the user. While only one service is sufficient for the data services, each functional service contains a specific name. Business services also contain a specific name for each service.

The section named "invoking services" describes in detail how to invoke Swoat services.

3.3.2.Swoat Deploy Methodology

This section describes the methodology that should be followed in order to use Swoat in software development. Several types of software development methodologies exist (based on Waterfall, Rapid Application Development, Spiral, etc.) like for example, eXtreme Programming (Baird, 2002), WIDSOM (Nunes, 2001), RUP (Silva and Videira, 2001) among others. Due the nature of Swoat framework deployment, it does not follow any of the mentioned methodologies but instead is based in the Semantic Information Management (SIM). This occurs because when Swoat is used in software development, the developer is not "implementing" a middleware but instead deploying and configuring one. The SIM methodology is useful in this case.

Detailing the Semantic Information Management (SIM) methodology (Bruijn, 2004a), methodology, the aim of this approach is to provide enterprises with insight into the information residing in different sources, in different formats, with different schemas across the enterprise. The SIM aims to provide a solution to this problem by creating a central ontology and mapping the individual source schemas to this central ontology, thereby creating a global view of all data residing in the organization (Alexiev et al., 2005) (Bruijn, 2004a) and (Bruijn, 2004b).

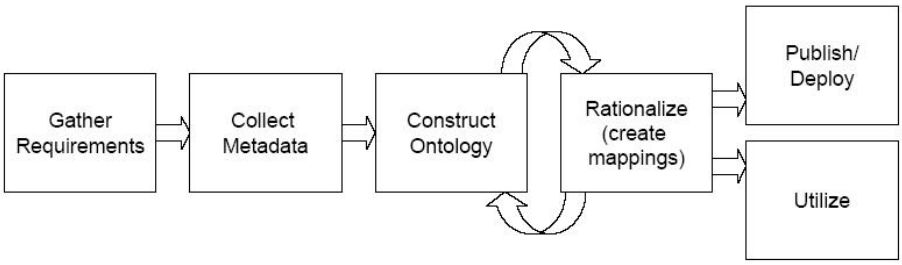


Figure 25 - The Semantic Information Management (SIM) methodology

As illustrated in Figure 25, the SIM methodology consists of six steps:

1. Gather requirements: the requirements for the information architecture are collected and the scope of the project established.

2. Collect metadata: all data assets relevant to the project are catalogued and an interface to access the data created.

3. Construct ontology: Create the ontology.

4. Rationalize: Establish the mappings between the data schemas and the ontology.

5. Publish/Deploy: The ontology, along with the mappings, is published to relevant stakeholders.

6. Utilize: Processes need to be created to ensure maintenance of architecture.

Swoat SIM based methodology differs from the original definition because the ontology created is not generated by 'reverse engineering' (following the LAV approach) the database schemas but instead generated from scratch (GAV approach) (Lanzerini, 2002), and then mapped to the database objects (table or view) that store the data described by the business entities (domain concept). This way, the created ontology, describes the 'to be implemented' and not the 'as is implemented'. Because the ontology is not generated from existing sources already created ontologies can be reused, and the created ontology can be distributed.

The drawback of the adopted solution is that mappings from the ontology to the database tables that store the data are created manually, and if changes occur in the database schema, the mappings have to be redone manually.

The following section describes how to map ontologies (information model) to the databases that store the data.

3.3.3.Establishing Mappings from the Ontology to Databases

The information model (ontology) implemented in OWL represents the classes, its attributes and relations between classes. Mapping the OWL model to the database is a fundamental step in which the main purpose is to establish a connection (mapping) between the ontology classes to the database tables that store the described data with the final objective of generating the SQL statement in order to get the required data.

In order to enable and ease ontologies reuse and distribution, Swoat separates the ontology from the mapping to the databases. This way the conceptual model (ontology) is completely separated from "implementation details" like the mappings that store the data location, which changes from organization to organization. How are the mappings implemented in Swoat framework? Mappings to the databases are implemented by creating ontology instances that contain the mapping to the database. This way, by detaching the ontology instances from the ontology, the "as is" ontology is obtained, completely independent from specific organization implementation details.

This section describes how to implement mappings from the ontology to the database. It first presents how to map ontology classes to database tables. It follows describing how to map the attributes of the ontology classes to the attributes of the database tables. One important issue in an information model is the relation between concepts. Therefore, it is explained how to map an ontology relation to a database relation.

Each section will start by presenting the mappings in a generic way. Then, it will follow with a real case (example).

3.3.3.1.Mapping Ontology Classes

When mapping ontology classes to database tables, the ontology instance name should have the same name as the database table that stores the data, preceded by the name of the database. Swoat uses the name of the instance to build the SQL statement in order to get the data.

The notation of an ontology instance name is {databaseName}.{tableName}:

- databaseName: Is the name of the database that stores the data.
- tableName: Is the name of the table that stores the data.

Because in some database systems the name of the database and of the tables is case sensitive, the ontology instance name should be the same as the name of the database and table.

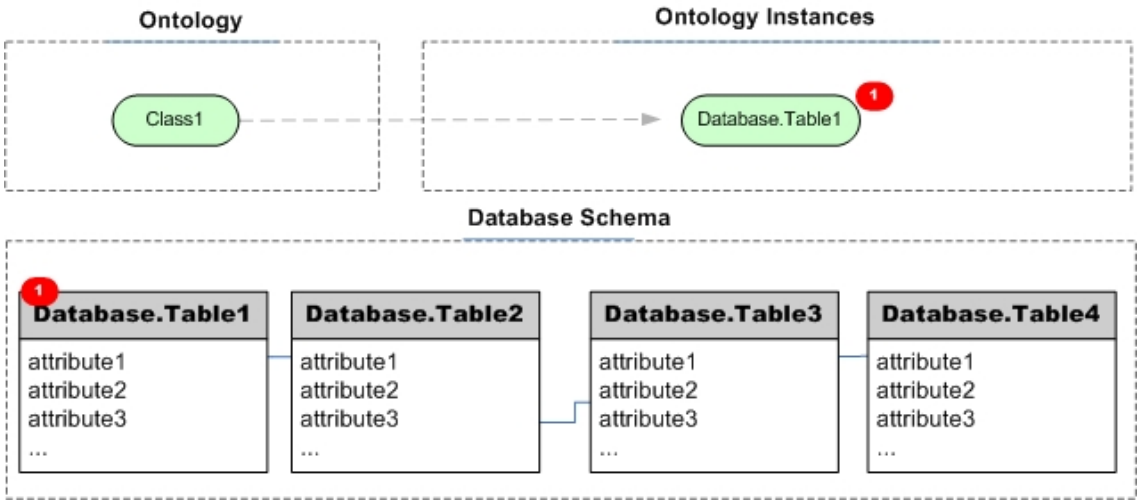


Figure 26 - Ontology instance names

Suppose that the ontology has a class that is named Class1 (illustrated at the top right corner of the Figure 26) that has its data stored in the table named "Table1" of the database named "Database". In this case, the name of the instance of the Class1 (illustrated in the top left corner of figure) should be Database.Table1 (illustrated in the top right corner of figure). This means that the name of the database that stores the data is Database and the name of the table is Table1 as illustrated at the bottom of the figure. The numbers illustrated in red (1 in this case) are used to relate the instance name with the database table. Summing up, the ontology class (top left corner of figure) describes data that is stored in "Database.Tabl1", that is the name of the instance (top right corner). The database table is illustrated at the bottom of figure.

For example, following a real case, suppose that the ontology contains a class named Person. In the organization systems, the person data is stored in the Human Resource database, named Personal, in a table named TablIdent (Ident from identification). In this case an instance of the class Person should be created with the name Personal.TablIdent (the name of the database should precedes the name of the ontology instance). This case is illustrated in Figure 27.

Examples of mappings are presented in the next table and illustrated in Table 3.2:

Ontology domain class name	**Ontology instance name and database table name**
Person	Personal.TablIdent
Contact	Personal.TablContact

Table 3.2 - Examples of instance names (database tables) of ontology classes

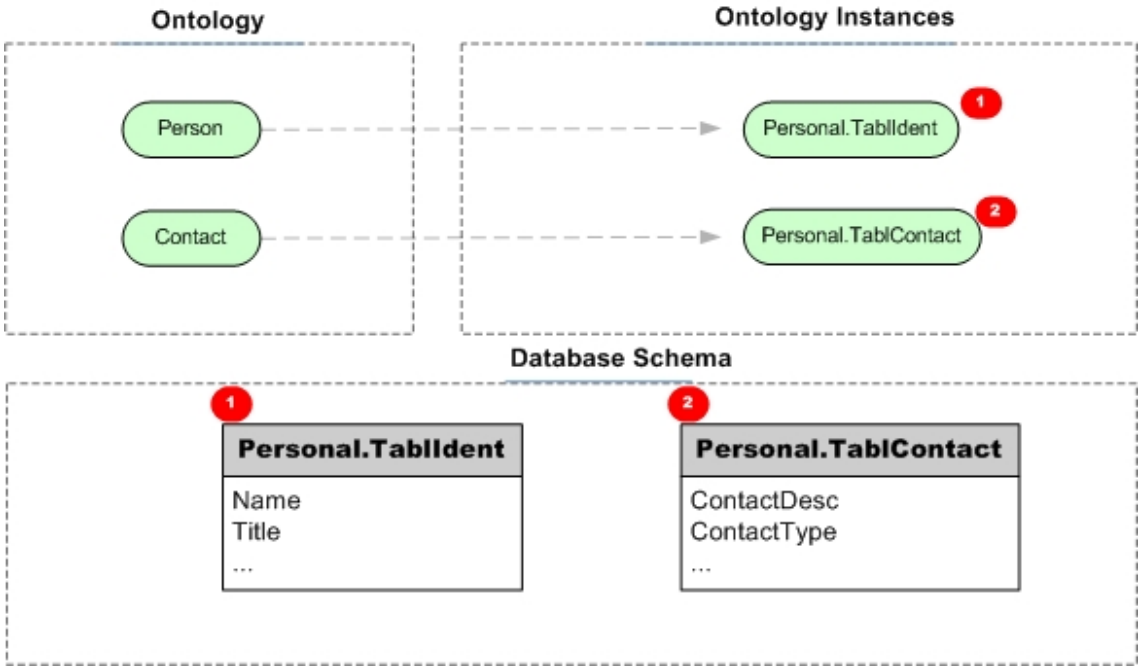


Figure 27 - Person and Contact ontology instances

This section described how to map ontology classes to database tables from a generic way. The next section describes how to map ontology classes attributes to database table attributes.

3.3.3.2.Mapping Ontology Attributes

When mapping ontology attributes to database table attributes, two cases may occur:

- The ontology class contains all attributes in a single table (the name of the instance).
- The ontology class contains attributes in several tables.

The main difference between these two types is that in the first case all attributes belong to the table that has the same name as the name of the ontology instance, while in the second case the attributes can be stored in several tables. Mapping ontology

attributes to database table attributes means that the value of the ontology instance attribute should contain the name of the table attribute that stores the data.

These two cases will be illustrated and exemplified in the next sections.

All Attributes in a Single Table

When all attributes are stored in a single table, the attribute of the ontology instance should contain the name of the table attribute. As a Swoat rule, all the mapped attributes should start by "field=" and should terminate with a semicolon (;). This is used in order to help building the SQL statement.

Therefore, the following notation should be used when creating mapping ontology attributes to table attributes: {*field=???;*}: The symbol ??? Represents the name of the field stored in the table (that is the same as the name of the ontology instance).

Figure 28 illustrates the same example presented in the previous section but complemented with the attributes. In this case the ontology class named Class1 contains two attributes named att1 and att2 (top left corner of figure). In this case, Class1 data is stored in the "Database.Table1" database table (as illustrated in figure at the top right corner) and field1 attribute contains the value "field=attribute1" and field2 contains the value "field=attribute2". This means that the ontology Class1 contains its data stored in the table "Database.Table1" and its attributes in attribute1 and attribute2 of the same table (Database.Table1).

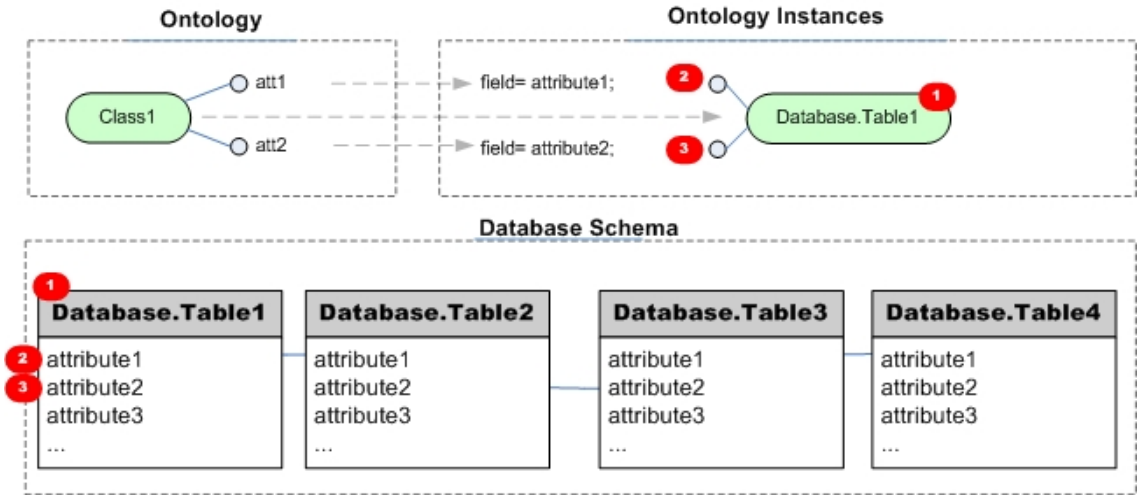


Figure 28 - Ontology attributes content

Summing up, the ontology class (top left corner of figure) describes data that is stored in "Database.Tabl1", that is the name of the instance (top right corner). The ontology class also contains two attributes, att1 and att2 that are mapped to the attributes of the

database table: attribute1 and attribute2. The database table and its attributes are illustrated at the bottom of figure.

Exemplifying with a real case, suppose that the ontology contains a class named Person with an attribute named name. This attribute contains the name of the Person. In the instance, these attributes will contain the name of the table attribute that store the data in the database.

The following picture describes the mapping of the class Person and of the attribute name:

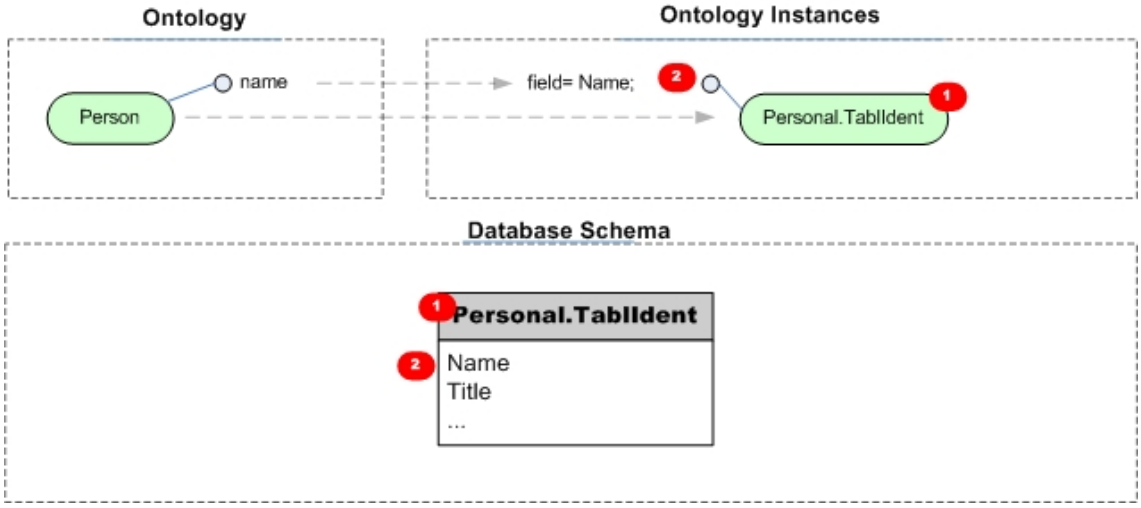


Figure 29 - Ontology instance attributes (name and title)

From the previous example, the person ontology class is stored in the database named Personal and in the table named TablIdent. The ontology attribute name is mapped to the table attribute named name, in the table TablIdent of the database Personal.

Attributes in Several Tables

In this case, the fields that belong to the table (the name of the instance) are mapped starting by field=???; as already presented in the previous section. The fields that do not belong to the table that has the same as the instance name, should be mapped using the following notation: table=???;field=???;path={???}:

- Table: The symbol ??? represents the name of the table that contains the specified attribute.
- Field: The symbol ??? represents the name of the attribute that is stored in the table (previously presented).

- Path: The symbol ??? represents the path to reach the table. The Path notation is: *field=???;table=???;field=???...field=???;table=???;field=???;*. It should always start and end with field=. At least one field attribute is required. The field (field=) represents the attribute that relates the table (the instance attribute name). The table (table=), represents the table name that is connected with the table that stores the attribute. This case is going to be illustrated more ahead.

Figure 30 illustrates the case in which the ontology class named Class1 contains data in a table that is not equal to the instance name. In this case, Class1 is mapped to the database table Table1 of database named Database. As depicted in the Figure 30, the attribute att2 is not stored in database table named Table1 (that is the same as the name of the instance), but instead, in Table2 and in the attribute2. At the top right part of the figure is illustrated the mapping that is required: table=Database.Tabl2;field=attribute2;path={field=attribute1}, detailed ahead:

- *table=Database.Table2* means that this attribute is not stored in "Database.Table1" that is the instance name but instead in Table2 of the database named Database.

- *field=attribute2* means that the ontology attribute named att2 is stored in the attribute2 of the table previously presented (Database.Table2).

- *path={field=attribute1}* means that the two tables (Table1 and Table2) are related by the attribute attribute1, as illustrated at the bottom of Figure 30.

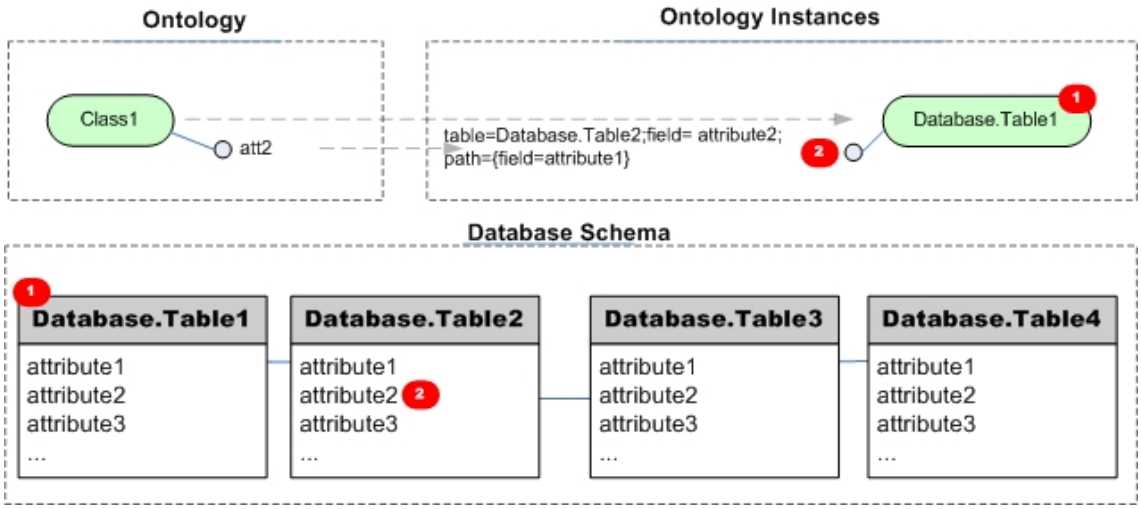


Figure 30 - Attributes in several tables

In the following example, illustrated in Figure 31, the ontology class Person (top right part of Figure 31) is mapped to database table named TablIdent in the database

named Personal (Personal.TablIdent). The title field is not contained in the table TablPersonal but instead in the database table named TablTitle, more precisely in the field named description. The relation from the table TablPerson to the table TablTitle is through the field idTitle, as illustrated at the bottom of figure and reflected in the attribute mapping depicted at the right top part of Figure 31:

- table=Database.TablTitle: the database and table that stores the ontology title attribute.
- field=Description: the name of the field in the table TabTitle that stores the ontology title attribute.
- path={field=IdTitle}: the relation between the table TablPerson and the table TablIdent.

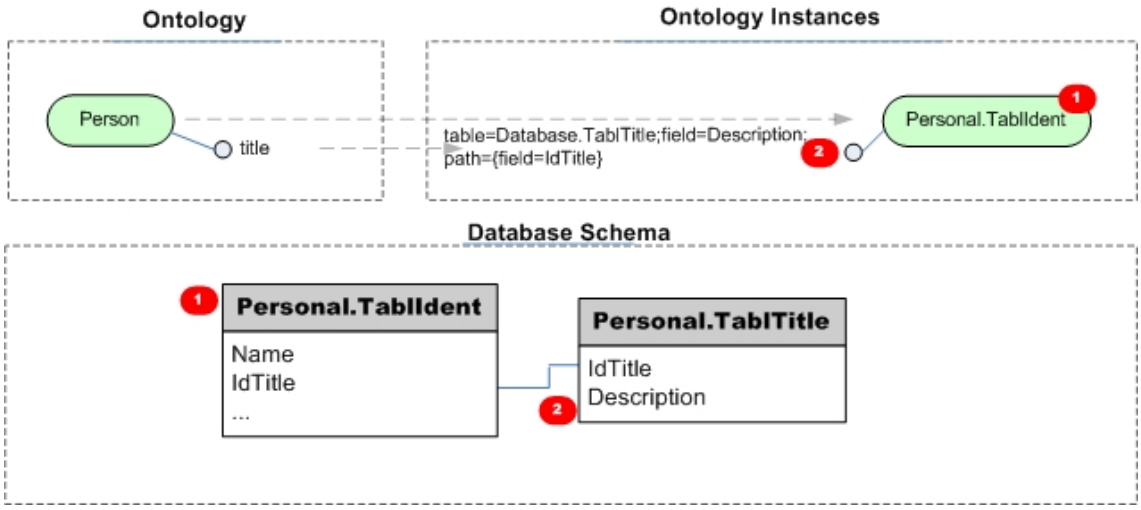


Figure 31 - Example of attributes in several tables

The following sections describe how to map relations from the ontology to the database.

3.3.3.3.Ontology Relations

While in the information model (described In OWL) relations connect domain concepts, in the database, relations connect tables. In order to map ontology relations to database relations, the chosen solution is to annotate the ontology relations. The content of the annotation property, that should have the name *relationAttribute*, should follow the following notation: *field=???table=???field=???...field=???table=????field=???*. It should start and end with 'field='. The number of 'tables=' depend on the number of intermediary tables that exist. The explanation is equal to the "*path=*" exemplified in the previous section.

Figure 32 depicts the example in which two ontology classes (Class1 and Class2) are mapped to Table1 and Table3 stored in the database named "Database". In the ontology the property that relates the two classes is named "hasProperty". In the database, Table1 and Table2 are related through an intermediary table, named Tabl2 and illustrated at the bottom part of the figure. The mapping is illustrated in top right part of the figure, leading to the conclusion that attribute1 relates Table1 with Table2 and attribute3 relates Table2 with Table3.

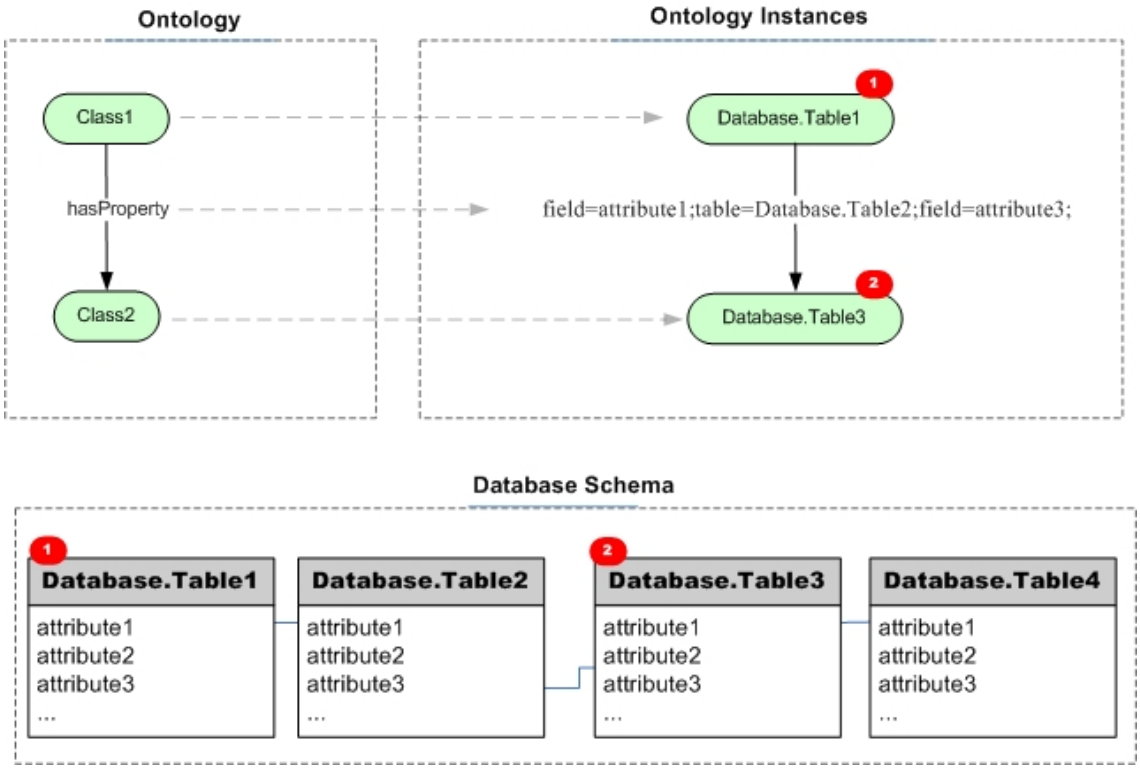


Figure 32 - Mapping ontology relations

Exemplifying, suppose that the information model describes two classes: Person and Contact, as illustrated in the top left part of Figure 33. These two ontology classes will be mapped to the tables TablIdent and TablContact, respectively. However, an intermediary table named TablPersonContact exists. Therefore, we need to store information about this table, because other way it will not be possible to generate the SQL to obtain the data. In this case, the property hasContact should be annotated with the following content in the relationField: *field=IdPerson;table=TablPersonContact;field=IdContact;*. The IdPerson is the field that connects the TablPerson to TablPersonContact. The IdContact connects the table TablPersonContact to TablContact.

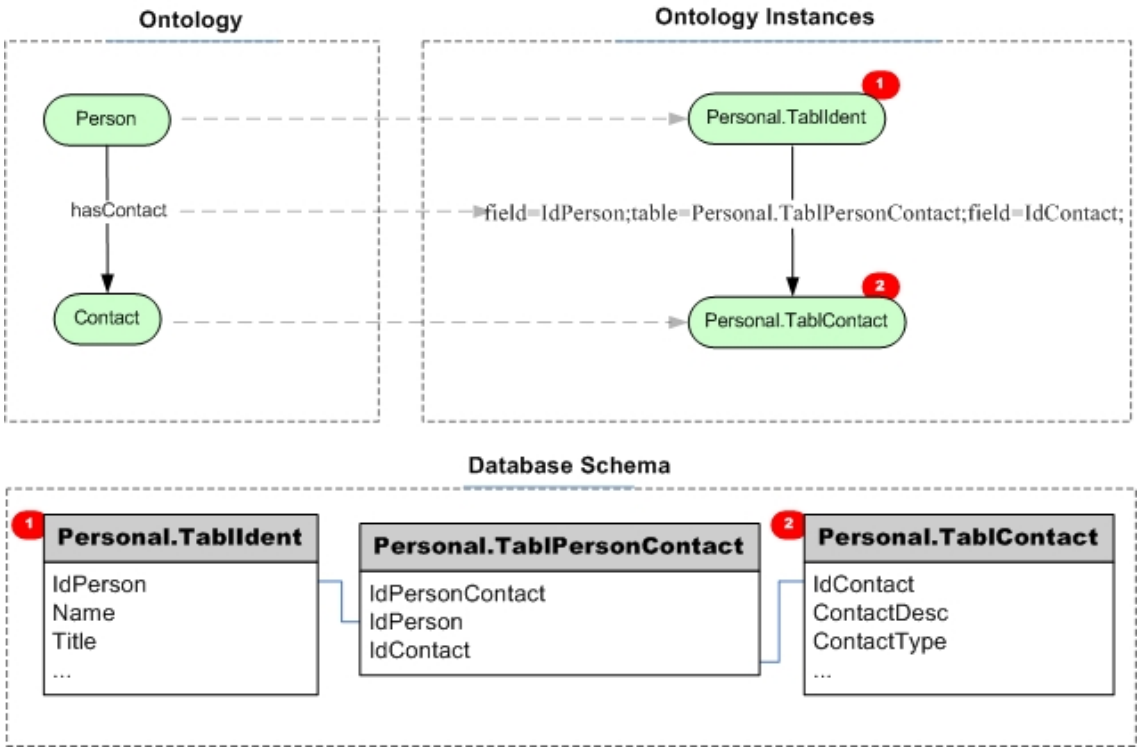


Figure 33 - Mapping relations (example)

Another case can occur when mapping ontology relations: when the name of the relation field is different between tables. Another case is when multiple relations occur between tables. These two cases are illustrated above in a generic way as described as "complex" relations:

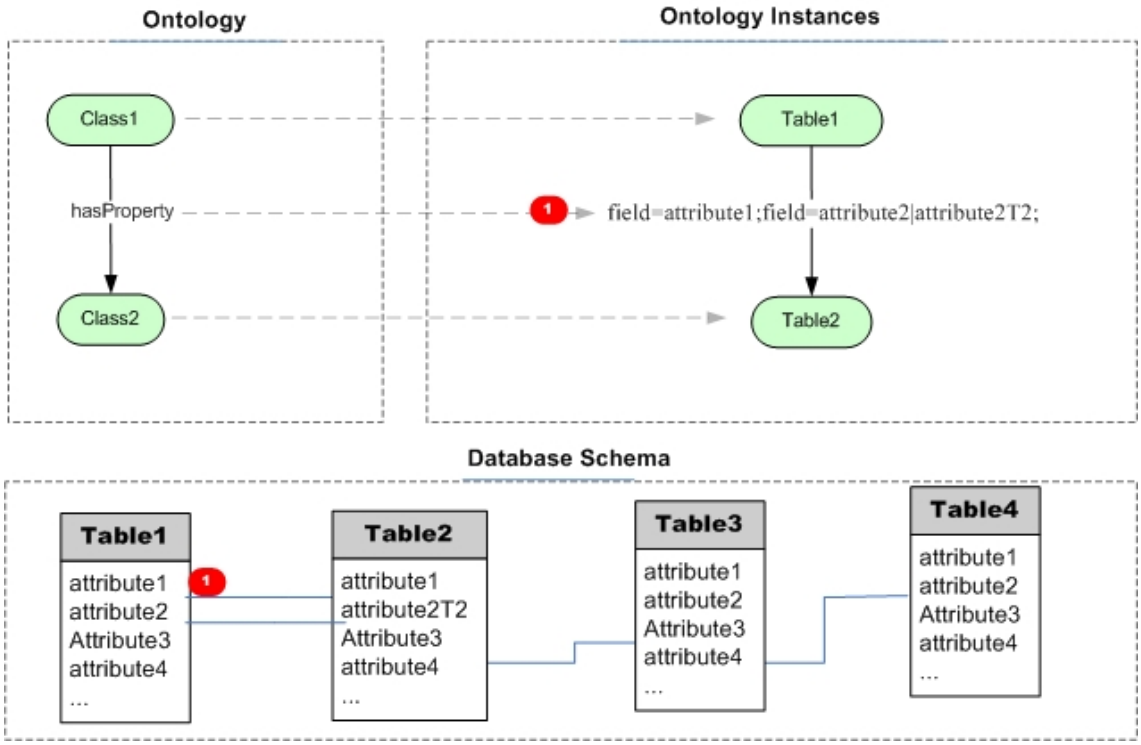


Figure 34 - Complex database relations

As illustrated in the bottom of Figure 34, Table1 is related with Table2 through the fields attribute1 and attribute2 (illustrated by two blue lines between Table1 and Table2). As also illustrated, attibute2 is stored in table1 but in table2 the corresponding relation attribute is named attribute2T2. The mapping in this case (more than one relation field) is achieved by separating the database relations with the character ";". The relation fields that are different in the relating tables are mapped using the character |. In Figure 34, at the top right corner, this mapping is illustrated: field=attribute1; that relates table Table1 with table Table2. The other relation is achieved by field=attribute2|attribute2T2; indicating that in table Table1 the relation field is named attribute1 and in Table2 it is named attribute2T2.

Until now the types of mappings have been presented separately: Ontology classes, ontology attributes and ontology relations. The following section combines all these types, and presents the four types of mappings that can occur.

3.3.3.4.Mapping Types

The previous sections described how to create mappings, focussing on how to map ontology instances to database tables, ontology attributes to database table attributes and ontology relations (between ontology classes) to database relations (between tables). This section summarizes and combines the mappings presented previously in order to present the types of mappings that can occur when mapping ontologies to databases. Four types of mappings can occur.

- An ontology class is mapped to one, and only one, database table.
- An ontology class is mapped to two or more tables in the database.
- Two or more classes of the ontology are mapped to one database table.
- A combination of the last two cases: two or more classes of the ontology are mapped to two of more database tables.

Mapping the ontology classes to the database tables, its attributes and relations is going to be described to all above-mentioned situations.

One Class of the Ontology Mapped to One Database Table

In this case, one class of the ontology corresponds to one table in the database. The procedure in this situation is to create an instance of the class with the same name of the table that stores the data. All the attributes should also be mapped.

Figure 35 illustrates the procedure for this situation, in which two ontology classes are mapped to two database tables (each ontology class to one database table), presented in a generic way. The ontology tables are related by one property. At the top left corner of the figure are illustrated the ontology classes and relations among them. In this case is illustrated the class class1 and the class2 and their attributes (att1 and att2), connected by the relation hasProperty. At the bottom of the picture is illustrated the database tables and relations among them. The ontology instances situated at the right top corner of the picture, store the mappings from the ontology classes to the database tables. The data described by the ontology class Class1 is stored in the table Tabl1, which is the name of the Class1 instance. The attributes are mapped by field=attribute1. As illustrated by the number (1), the att1 attribute of the class Class1 is stored in the attribute of the table Table1. The same happens to (2). Attributes (3) and (4) are stored in Table2 in attributes 1 and 2 respectively.

As depicted in the ontology, the Class1 is related to the Class2 though the relation hasProperty. This is reflected in the database by the connection between Table1 and Table2 (which in this case has no intermediary relation table). The field that relates the two table is attribute1 as illustrated by (5) in the ontology instance.

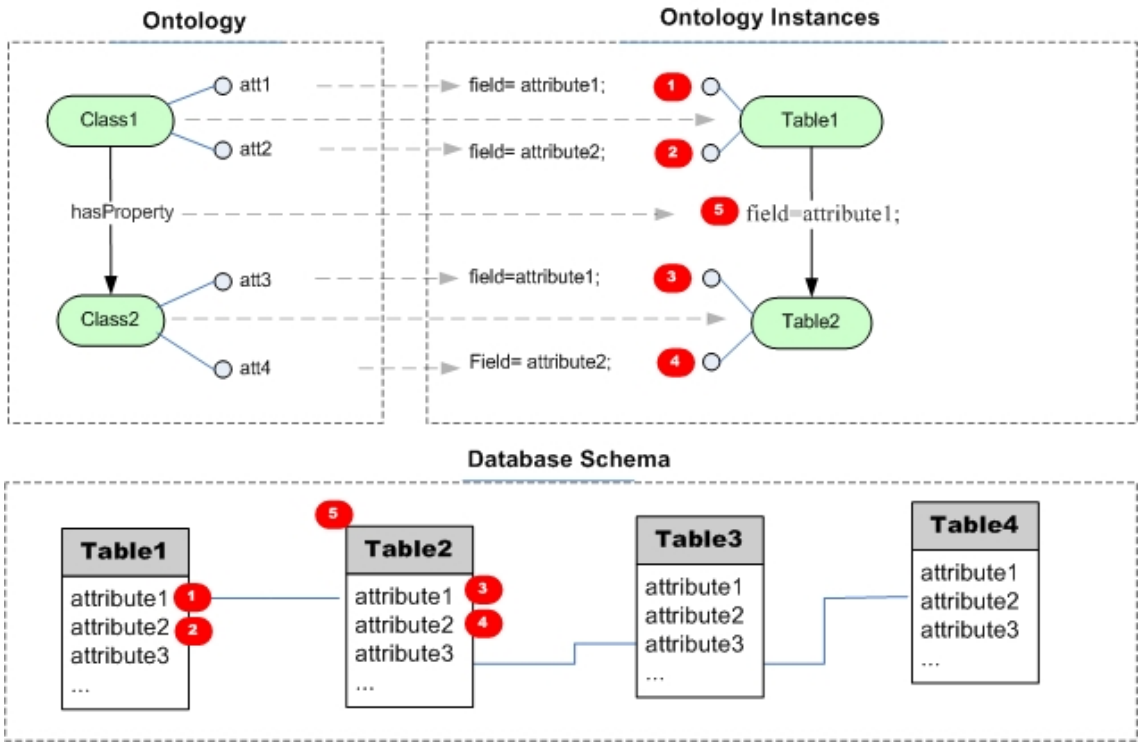


Figure 35 - One class of the ontology and one database table

One Class of the Ontology and Two or More Database Tables

In this case one class of the ontology describes data that is stored in two or more tables, as illustrated in Figure 36. The name of the ontology instance should have the same name as one of the tables. All the ontology class attributes stored in the table which the name is the same as the instance name should be mapped as already presented (One class of the ontology and one database table).

Figure 36 illustrates the ontology class Class1 (top left corner) that contains two attributes. The ontology attribute named att1 is stored in table1 more precisely in attribute1 and att2 is stored in Table2, in attribute 2. The two tables are related by attribute1, as illustrated in bottom of the figure and mapped in the attribute that do not belong to the Table1 (path={field=attribute1;};).

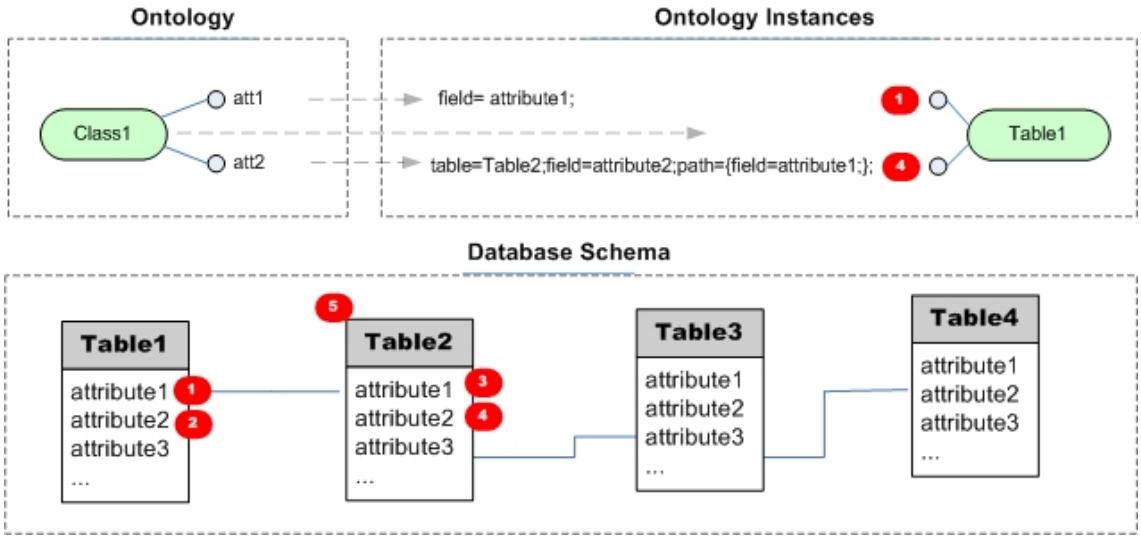


Figure 36 - One class of the ontology and two or more database tables

Two or More Classes of the Ontology and one Database Table

In this case one or more ontology classes describe data that is stored in the same table, as illustrated in Figure 37. This situation leads to the creation of several ontology class instances with the same name. By creating ontology instances this would be a problem because of one characteristic that is UNA (unique name assumption). In order to solve this problem the mappings should be addressed naming the instances with the symbol "_".

In the following figure, Class1 and Class2 (top left corner of the picture) both describe information that is stored in Table1. The instance (top right corner) should contain the same name, but because this is not possible, one of the instances was renamed starting by the symbol "_".

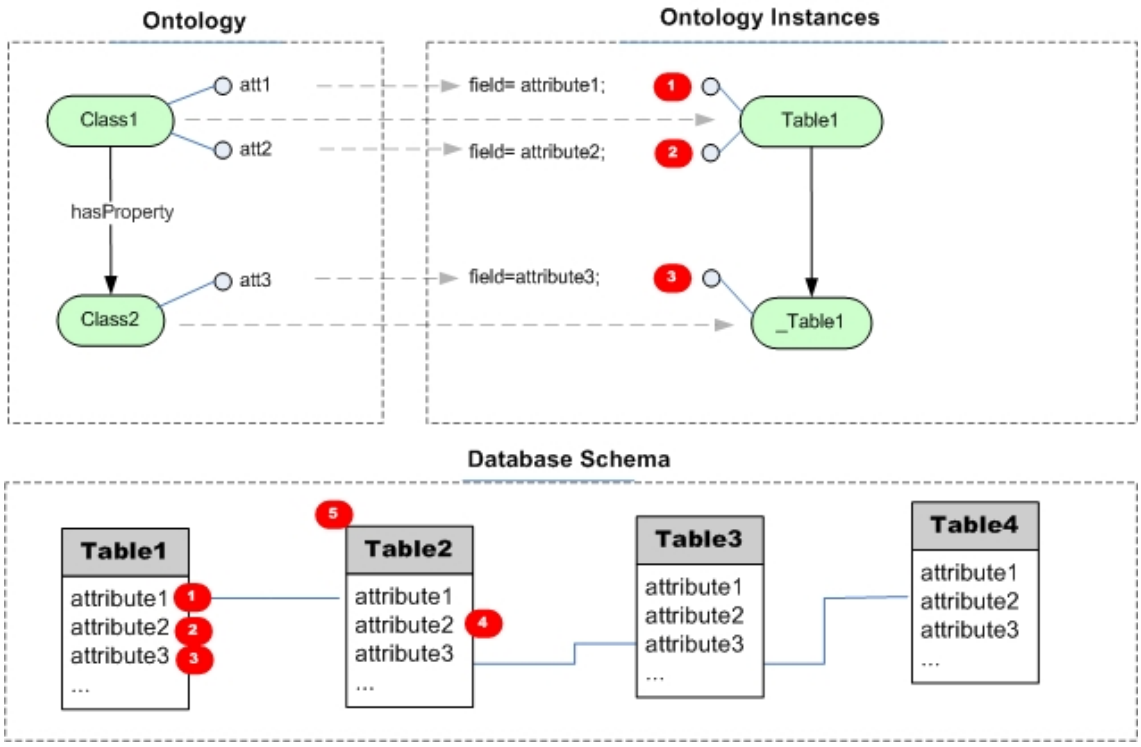


Figure 37 - Two or more classes of the ontology and one database table

Two or More Classes of the Ontology and Two or More Database Tables

This case is the combination of previously presented cases. The procedures presented above should be followed.

Two or More Classes of the Ontology with Specialization/Generalization

In information models (ontology) it is common to find specialization (referred as sub class)/generalization (referred as super class) relations. In database models it is not so evident, because it is not a pre-existing database relation. This section describes how to map ontology specialization / generalization relations to database.

The specialization/generalization procedure is to map the attributes that belong to each specific case, i.e. the inherited attributes should not be mapped in the son class. However, one special procedure is needed, that is the creation of a property in the ontology that relates the two ontology classes (the super-class and the sub-class). This property should only be created in the case when the ontology father/son classes are stored in two database tables. This "special" relation is needed because of the annotation property that should be created and is also motivated by the fact that in databases the specialization / generalization are achieved through table relation. As already explained, table relation mappings are achieved by creating an annotation property in the relation.

The procedure to follow in this case is the same as presented in "One class of the ontology and one database table".

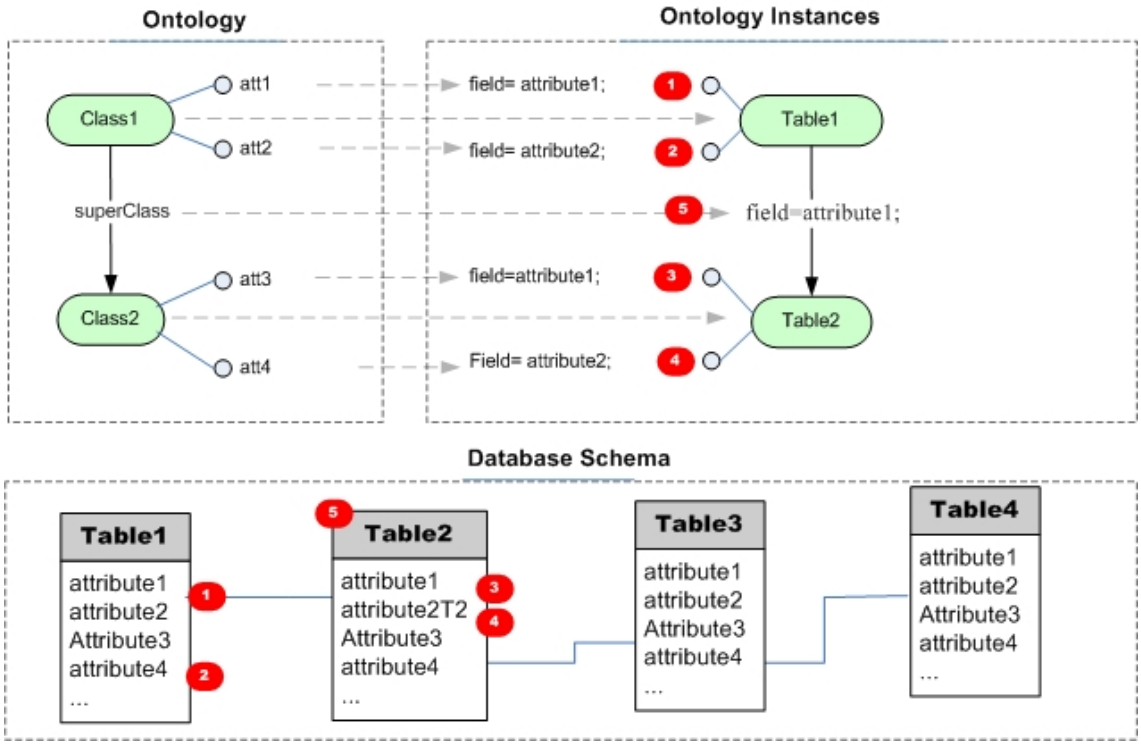


Figure 38 - Ontology specialization/generalization example

As illustrated at the top left corner of Figure 38, Class1 is the super class of Class2, meaning that Class2 is a specialization of Class1. Also, Class2 contains Class1 plus Class2 attributes because Class1 attributes are inherited from Class1. As illustrated at the top right of figure, all the Class1 attributes should be mapped in the Class1 instance and the Class2 in class2 instance. Class2 inherited attributes from Class1 should not be mapped, because they were already mapped in Class1.

3.3.4.Invoking Services: Service Request and Response

Swoat offers three types of services: data, functional and business. Data services essentially return data from the database. The other two types of services, functional and business, use the data service, since usually data is needed (the basic "ingredient") in order to build a service. Due to this issue, Swoat focused on the implementation of data type service that is detailed in this section.

Clients interact with Swoat by invoking services, implemented using Web Services, in which the message content of the request and response is structured in XML and encapsulated in a SOAP envelope. Also, clients interact with Swoat making service requests (which contains 'what data' is needed), using a XML structure named NCQL (Neutral Client Query Language) defined by the XSD schema illustrated in Figure 39. Internally, Swoat transforms this request language in order to build a SPARQL (an "SQL" language to OWL) query. Several reasons motivated the development of a query language (NCQL), instead of using SPARQL (the query language for ontologies):

- Clients specify requests using simple XML syntax. This way, the request query language is implemented independently, which means that even if implementation technologies of the Swoat framework and database changes, clients don't need to be changed.

- Clients are abstracted from SPARQL (being aware of its existence) leading to a faster learning curve of users.

- It is easier to build automatic requests from "front-end" tiers.

Since Swoat intends to be used as a building block in the development of applications (the middleware) leading to a three tier approach, one major concern is the build of requests (by clients). In the case of data services, requests contain what data is needed. Usually in two tier application development, this type of requests is implemented using SQL queries, that contains what information to return and where is stored. NCQL does not differ abruptly from SQL, but focus essentially on specifying what information is needed. Thus, the above mentioned three reasons that motivated the development of NCQL are oriented towards the creation of a solution that does not differ substantially of SQL. Consequently, interface developers are also familiar with the result of an SQL statement: a record set. Suitably, other NCQL requirement is that results should also be quite similar to the result of SQL queries.

NCQL allow users to specify:

- Fields (ontology attributes) to be returned, using the *outputFields* element. This XML element can contain any ontology class attribute that are intended to be returned. The attribute name should be "case sensitive".

- The order of the output fields (ascending- ASC, descending- DESC), using the *orderFields*. The name of the attribute should be specified and the type of ordering: ASC or DESC.

- Filters (for example, return only names started by letter A), using the *filters* element.

- Choose the path that connects domain classes, using the *outputProperties* element.

The XML elements are described in more detail in the Figure 39 that illustrates NCQL XSD Schema. It describes each element and the attributes that can be specified (dashed rectangles represent optional XML elements and all the others are required).

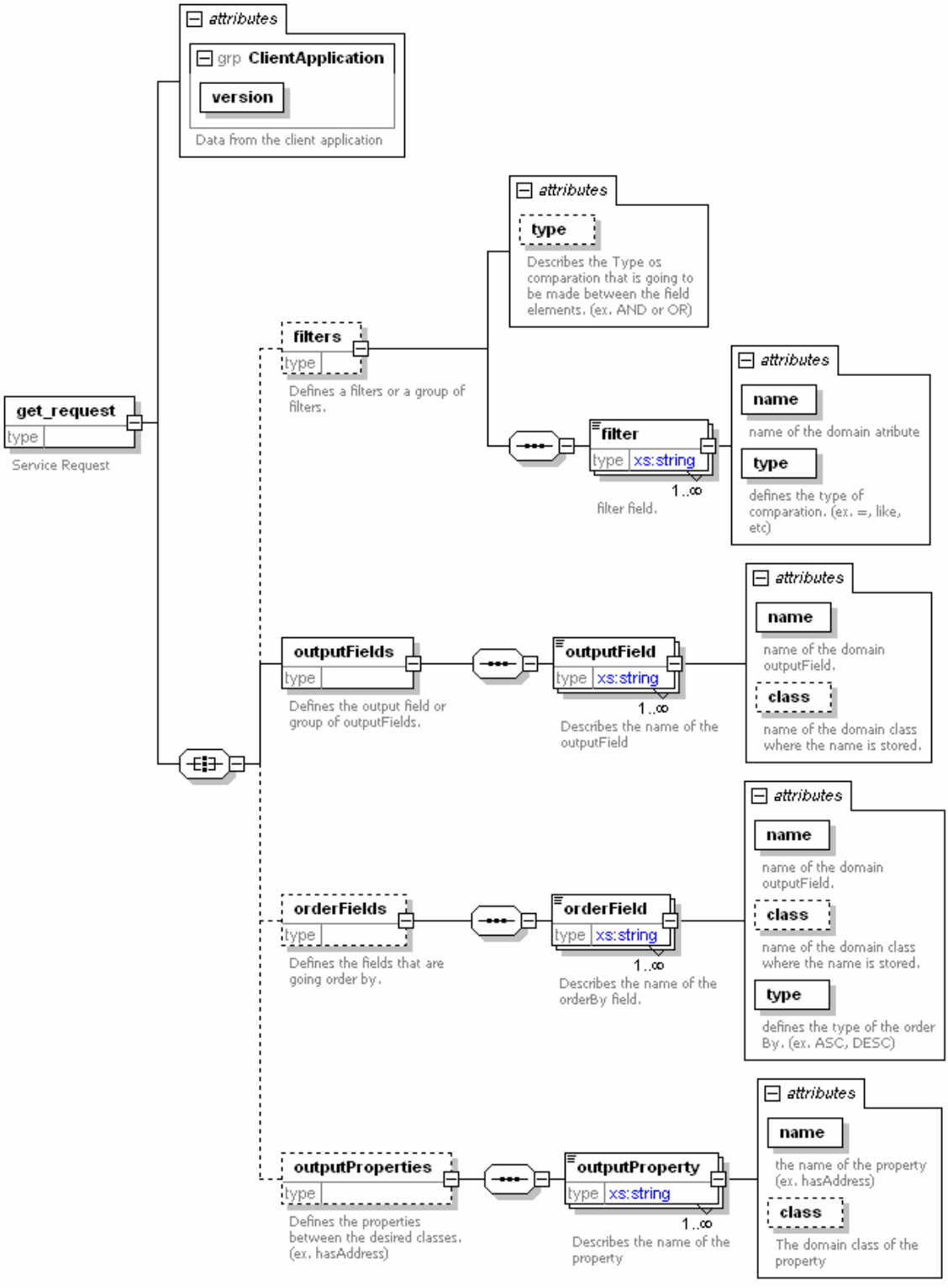


Figure 39 - Schema of the XML request

The NCQL request should always start with a XML element named *get_request*. This is the root element of NCQL. *get_request* contains an attribute that is named *version* that stores the version of the request that clients are invoking. This allows that even if

NCQL changes, old clients can continue using the previous version of NCQL. The following XML element illustrates the element get_request and its attribute.

```
<get_request version="1.0"></get_request>
```

The get_request element can have 4 child elements: filters, outputFields, orderFields and outputProperties.

The filters element contains an attribute that specifies what type of filter is desired: AND or OR. The filters element can have a child element named filter. This element specifies the name of the ontology attribute to filter and the type of filter, which can be > (greater than) < (less that) or = (equal). These types can also be combined: >=, <= or not equal.

For example, the following case describes two filter elements (name equal to John and age equal to 99) that should be verified:

```
<get_request version="1.0">
<filters type="AND">
        <filter type="=" name="name">John</filter>
        <filter type="=" name="age">59</filter>
</filters>
</get_request>
```

The *outputFields* element contains the elements that should be returned. Since several elements can be returned, *outputFields* contains a child element named *outputField* that describes the name of the attribute and the name of the class that contains it, as illustrated:

```
<get_request version="1.0">
 <filters type="AND">
        <filter type="=" name="name">John</filter>
        <filter type="=" name="age">59</filter>
 </filters>
 <outputFields>
        <outputField name="name" class="Person"/>
        <outputField name="age" class="Person"/>
        <outputField name="description" class="Address"/>
 </outputFields>
</get_request>
```

The *orderFields* is similar to the *outputFields* element. In this case, the *orderFields* contains a child element named *orderField* that has three attributes: name (of the attribute to order), class (that stores the attribute) and type (ascending or descending). An example, which sorts ascending the name of the class person, is illustrated:

```
<get_request version="1">
        <filters type="AND">
                <filter type="=" name="name">John</filter>
                <filter type="=" name="age">59</filter>
        </filters>
        <outputFields>
                <outputField name="name" class="Person"/>
                <outputField name="age" class="Person"/>
        <outputField name="description" class="Address"/>
        </outputFields>
        <orderFields>
                <orderField type="ASC" name="name"/>
        </orderFields>
</get_request>
```

The *outputProperty* described the path between classes. This is usually not required, but enables the programmer to choose the path between classes, since circular relations can exist between classes. This element can contain several child elements named *outputProperty* that have two attributes: name (of the attribute) and class (that stores the attribute).

An example that relates the class Person with the class Address through the hasAddress property:

```
<get_request version="1">
        <outputFields>
                <outputField name="name" class="Person"/>
                <outputField name="age" class="Person"/>
                <outputField name="description" class="Address"/>
        </outputFields>
        <orderFields>
                <orderField type="ASC" name="name"/>
        </orderFields>
        <outputProperties>
                <outputProperty name="hasAddress"></outputProperty>
        </outputProperties>
</get_request>
```

The outputProperties will bias the response of the service. This will be described more ahead in this section.

The table compares SQL with NCQL:

Functionalities	**SQL**	**NCQL**
Specify fields to return	yes	yes
Filters (restriction conditions)	yes	yes
Ordering	yes	yes
Grouping	yes	yes
Functions (count, average, sum, …)	yes	no
Left Join, Right Join, Cross Join	yes	yes
Focus on	*what* information is needed and *where* it is stored	*what* information is needed
Database identification	required	not needed
Table identification	required	not needed
Database Abstraction	no (ex. if a database table changes the SQL also has to be changed)	yes (NCQL does no depend on the database vocabulary)

Table 3.3 - SQL operations compared to NCQL

Swoat processes NCQL by transforming it in a SPARQL query in order to get the information to build the SQL statement. To accomplish it, Swoat first reads the OWL file manipulated by protégé (OWL editor). Then, using the Jena API (java framework for building semantic Web applications) and the ARQ API (SPARQL query engine) the generated SPARQL query is executed, returning the result in XML format. Using XSLT stylesheets, applied to the result of the SPARQL query, an SQL statement is obtained in order to get the required data. After that, the result of the SQL query is transformed in a XML format, reflecting the ontology structure.

Due the technical nature of these transformations and SQL/XML generation, this section is not going to present it in detail. Instead it will focus in detail in the request

and response XML structures. The following picture, Figure 40, illustrates in a generic way the request (left part of figure), the service response (right part of figure) and its relation with the ontology (illustrated in the centre).

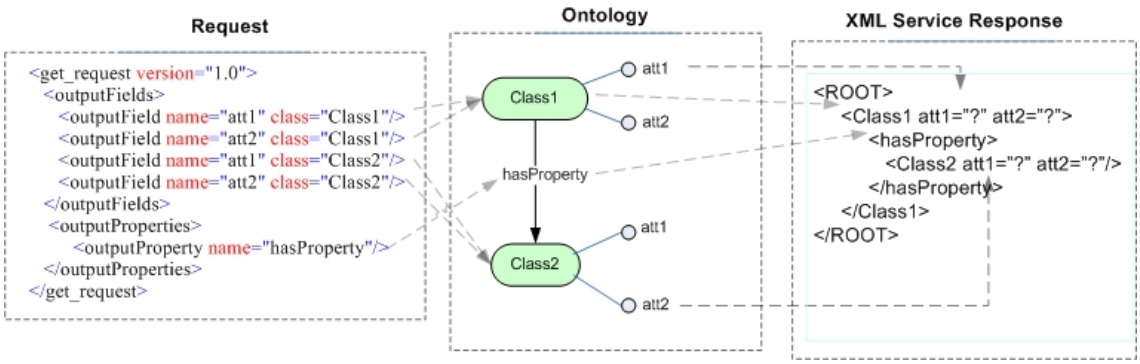


Figure 40 - Relation between request, ontology and XML service response

Figure 40 arrows between request and ontology illustrate existing relation between the request and the ontology. The arrows between the ontology and the response illustrate the relation between the response and the ontology.

For example, in order to get the name and address information, the request would be structured like:

```
<get_request version="1.0">
  <outputFields>
       <outputField name="name" class="Person"/>
       <outputField name="address" class="Address"/>
  </outputFields>
   <outputProperties>
         <outputProperty name="hasAddress"/>
  </outputProperties>
</get_request>
```

In the output fields, all required fields and the classes, which contain the attribute, are specified. In this particular case, we are interested in the name of the person and in its address. The output property is hasAddress, which means that the class Person and Address are related by hasAddress. The data *returned* would be:

```
<Root>
<Person name="John Doe">
        <hasAddress>
                <Address address="Statue Avenue"/>
        </hasAddress>
</Person>
</Root>
```

As already depicted the information model, the class Person is connected with the class Address through the relation hasAddress.

3.4. PERFORMANCE RESULTS

This section focuses on performance issues about Swoat. No exhaustive performance tests have been done, but the following results have been obtained in the above presented application.

The test scenario was a software application that invokes three Swoat data services in order to load the screen. In this case, the three services were used to load a "combo-box", a table and some screen fields, as illustrated in Figure 41.

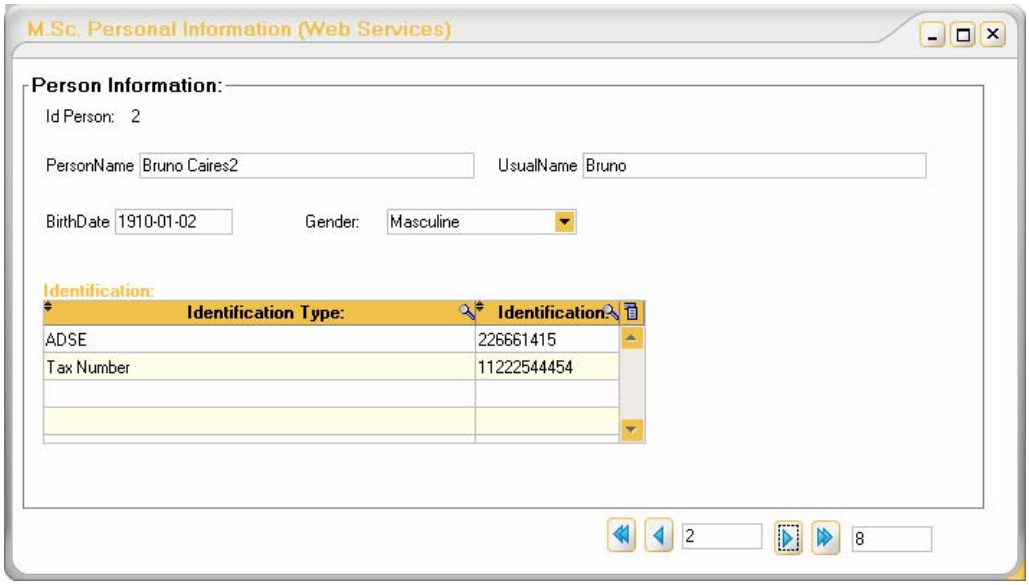


Figure 41 - An example application that invokes Swoat services

Invoking three SWOA data services, the time to load the "front-end" application varies from 0.2 to 0.6 seconds, which takes an average of 0.06 to 0.2 seconds per service. Using direct access to databases, the time to load the screen varies from 0.01 to 0.04 seconds. Figure 42 illustrates Swoat performance results of the three services invoked.

The tests have been accomplished using the following scenario: Pentium III 800MHz with 256 Mb RAM. The computer was running Windows XP, Sun Web Application Server 9.0 and Java 5.0 build 09. Tests in a server, running Linux have not been made, but it is expected that the time to load the screen decreases several times.

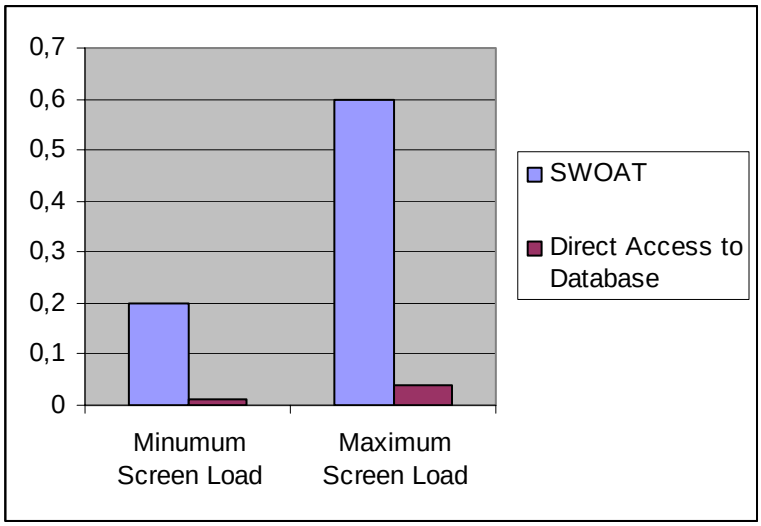


Figure 42 - Swoat performance results

The performance Swoat results are several times lower than database direct access. However, it was easily noticed that the development time of the application was faster than recurring to SQL queries, also taking advantage of (1) "front-end" depend on the information model and directly of the database schemas; (2) The services can be invoked even by heterogeneous applications, allowing integration with other systems.

3.5. CONCLUSION

Swoat (Service and Semantic Web Oriented ArchiTecture) framework is a middleware that when deployed between database and "front-end" clients, the system (database + middleware + "front-end") takes advantage of the following issues:

- The request done by clients, formulated to get the required data, focus on *'what information is needed'* and the less possible about 'how' information is obtained. 'How' related aspects like database location and SQL statements are transparent to the clients.

- *Changes that occur in the database* are not necessarily propagated to all clients. In this way, clients are not aware of the database changes, either syntactic (ex.: change of a table name) or structural (added or deleted table).

- Hides the local databases vocabulary, providing a *common vocabulary* across several databases. This way semantic heterogeneity is solved.

- Allows reusing of implemented application functions.

These issues are the Swoat requirements. In order to allow the above mentioned objectives, the following technologies were combined: Enterprise Information Integration (EII), middleware, Service Oriented Architecture, Web Services and Semantic/Ontologies.

Using *EII and middleware*, "front-end" clients remain decoupled from databases (database and "front-end" are mediated by Swoat). According to EII, the data remains in the original sources, being provided to clients a global and virtual view over the entire set of databases. Therefore, the combination of EII and middleware allows that databases changes that occur (change of a field, table, etc) don't be propagated to all clients connected to databases (that can be several), thus generating an enormous amount of maintenance.

The global view (domain model, information model and ontology– considered synonyms) is stored and centralized in the middleware. This model is formally specified using semantic (OWL, a W3C standard to describe ontologies) thus providing and accurate representation of the domain of the organization. Since

information models usually don't contain any specific organization information, it is able to be shared and or reused across several organizations. Therefore, *Semantic Web* mainly using ontologies and formal languages to specify ontology provides a viable technology to be used as a solution for describing the information model.

Other technology used, *Service Oriented Architecture* (SOA), allows that clients interact with Swoat by invoking services. Consequently, the services implemented especially to be called from a specific the "front-end" application can also be invoked by other applications, allowing function reusability. Adopting Web Services, language independence was achieved.

This chapter also reported the methodology used to embrace Swoat in a real case application. Swoat methodology is based on SIM, but differs in a way that the global model (ontology) is built independently from the existing databases. This way, even if the database schema changes the model remains (stored and centralized in the middleware) the same avoiding that clients have to be changed.

One fundamental stage in the methodology is mapping ontology classes (described by the model) to the database tables. The mappings are stored in the ontology instances allowing reuse and distribution of the ontology. This means that at any time the mappings can be detached, therefore obtaining the information model independent of technical and organization specific issues. Also, it is possible to reuse an already implemented model.

After the mappings stage, clients can invoke Swoat services, that are exposed using open standards, more precisely, using WS. The neutral client query language (NCQL) is the language used by clients to specify what data is needed. NCQL was developed in order to make the language that clients use to invoke services completely independent from the middleware technologies. Suitably, even if changes occur in the middleware (like changing the formal language - OWL) clients may not have to be changed.

4. SWOAT RUNNING EXAMPLE

> "Argument is conclusive, but it does not remove doubt, so that the mind may rest in the sure knowledge of the truth, unless it finds it by the method of experiment."
> — Roger Bacon (1214-1294)

This chapter starts by illustrating the environment in which Swoat is applied, focussing on the presentation of the ontology, which represents the global information model deployed in Swoat. It follows with the schema of the database that shows the tables that store the data. Mappings from the ontology to the database tables are described presenting the instances of the ontology. A small application that invokes Swoat services is also illustrated, detailing Swoat service request and response.

4.1. SWOAT DEPLOY ENVIRONMENT

Suppose that there exists an organization, a university in this case, with a system to manage human resource information, focussing on the personal data of their employees and main stakeholders (students). Therefore, information about the person (employee or student) like the name, birth date, address and contact is stored on the human resource database.

Actually there exists an application developed in java that allows internal employees to introduce and update information of the employees stored on the system. This application was bought from an external (it is a COTS system) organization and therefore it can be changed by the organization that developed it at any time. The database of the application is relational developed in MySql and it is possible to read data directly from tables.

In a short time span, it is expected that employees, properly authenticated, can also change some part of the information (like contact and address) using the organization Web site already implemented, using internal technical human resources. Also, it is intended that anyone can visualize the contact information of every employee, more precisely the email and internal phone number, on-line, using the organization site even without being authenticated.

This is the scenario in which Swoat is going to be applied. It is intended to abstract developers of the Web site from aspects like where is located the data, in which tables and attributes. Also, since the database can change at any time, Swoat intends to decouple (minimize the propagation of changes) when changes occur in the human resource system.

4.2. ONTOLOGY

The following figure, Figure 43, illustrates the ontology classes and relation among them. Since this figure represents the ontology in a high level, it only illustrates classes (the attributes will be shown in the next section). In the centre of Figure 43 is the class Person, represented in green. The connections (blue lines) between classes describe existing relations between classes.

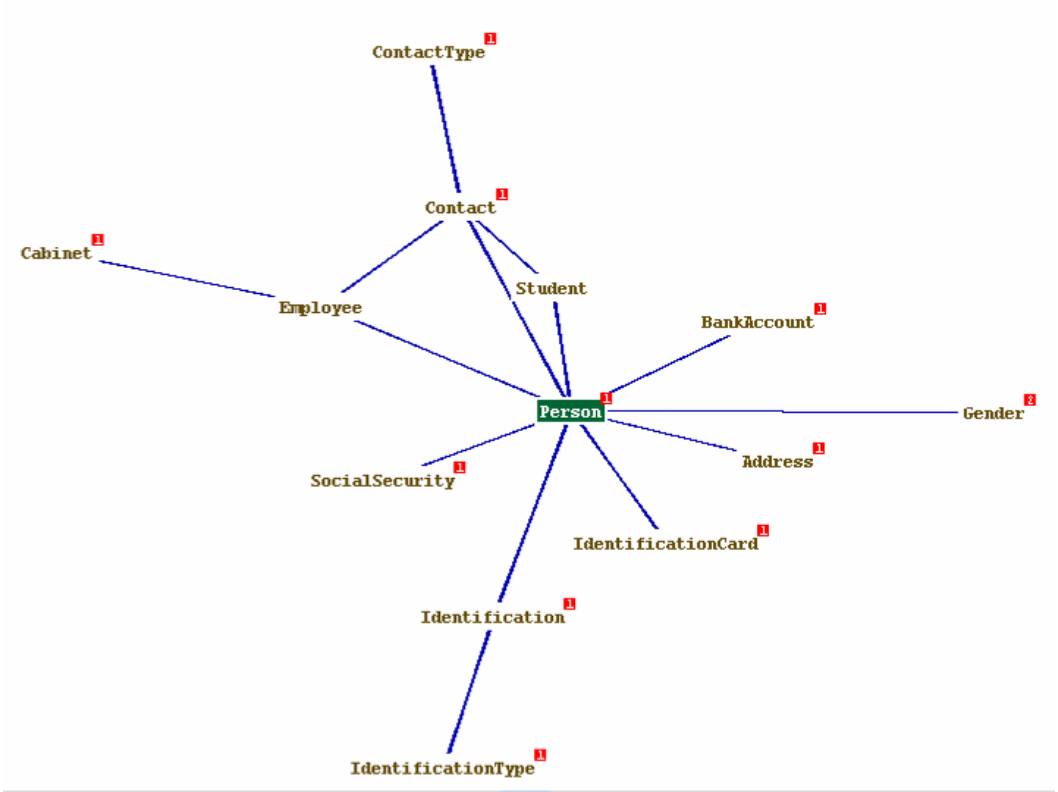


Figure 43 - High level representation of the ontology

As illustrated in the Figure 43, the ontology describes information like the person contact and the address. It also describes its identifications like the social security number, the identity card, among others. This ontology was built from scratch and intends to describe the "personal information".

The following picture illustrates the ontology concepts and its relations in more detail.

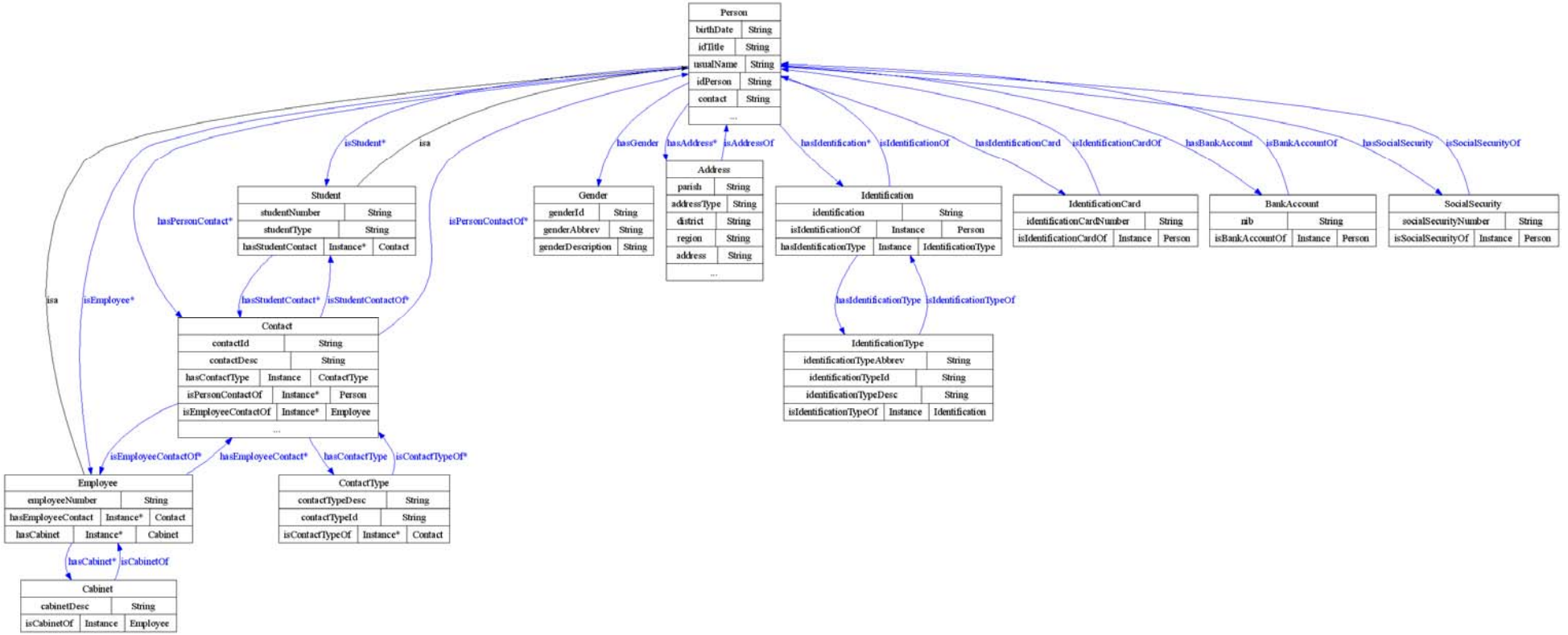


Figure 44 - Detailed representation of the ontology

The blue lines between the ontology classes represent the properties and the black lines the super-class/sub-class, describing that the subclass inherit the attributes from the super-classes.

Classes are depicted having in the top the class name and above the attributes.

4.3. EXISTING DATABASE

The scenario in which Swoat is going to be applied contains only a database. This database is relational, and contains personal data about organization employees. Example of data that the database store is: address, contact, the name of the person, birth date, among others.

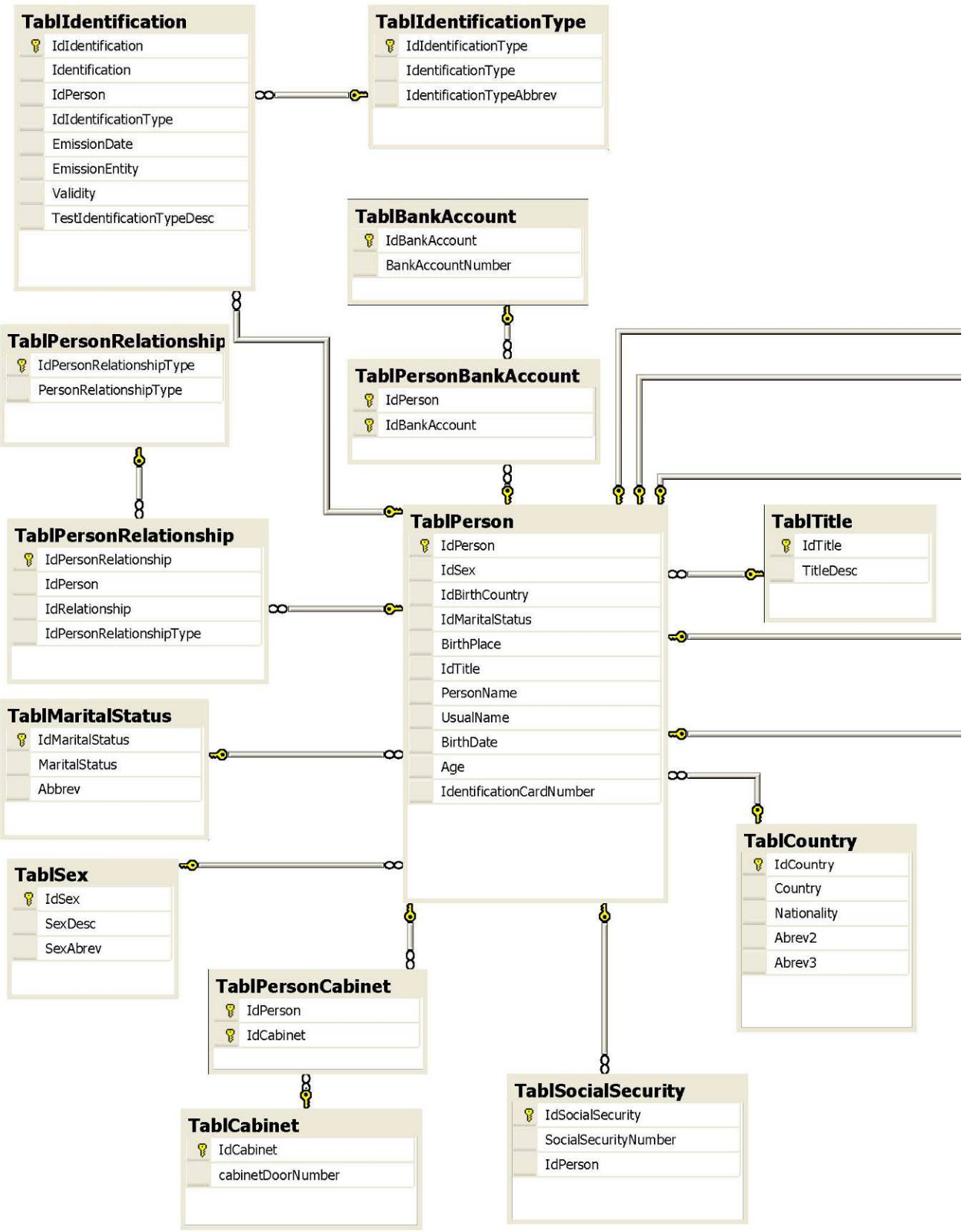
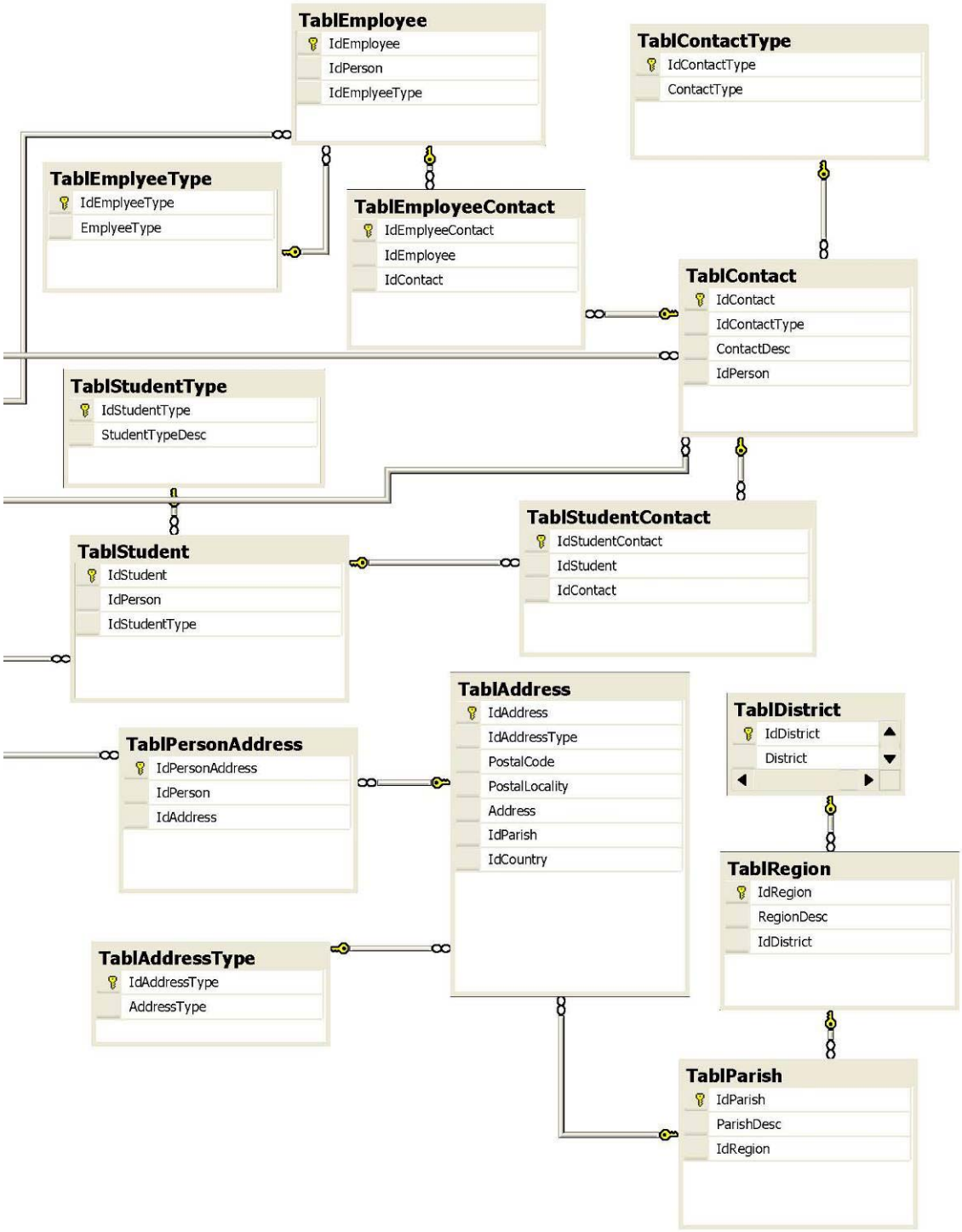


Figure 45 - Database schema (part 1 of 2)



1-2

Figure 46 - Database schema (part 2 of 2)

All the tables start by Tabl indicating that this type of object is of table type.

4.4. MAPPING – ONTOLOGY INSTANCES

Previously the database schema was presented. Also, the ontology created was illustrated. The next step is to map the ontology classes to the database tables. Therefore, the following picture shows the main ontology instances that store all the mapping information to the relational database.

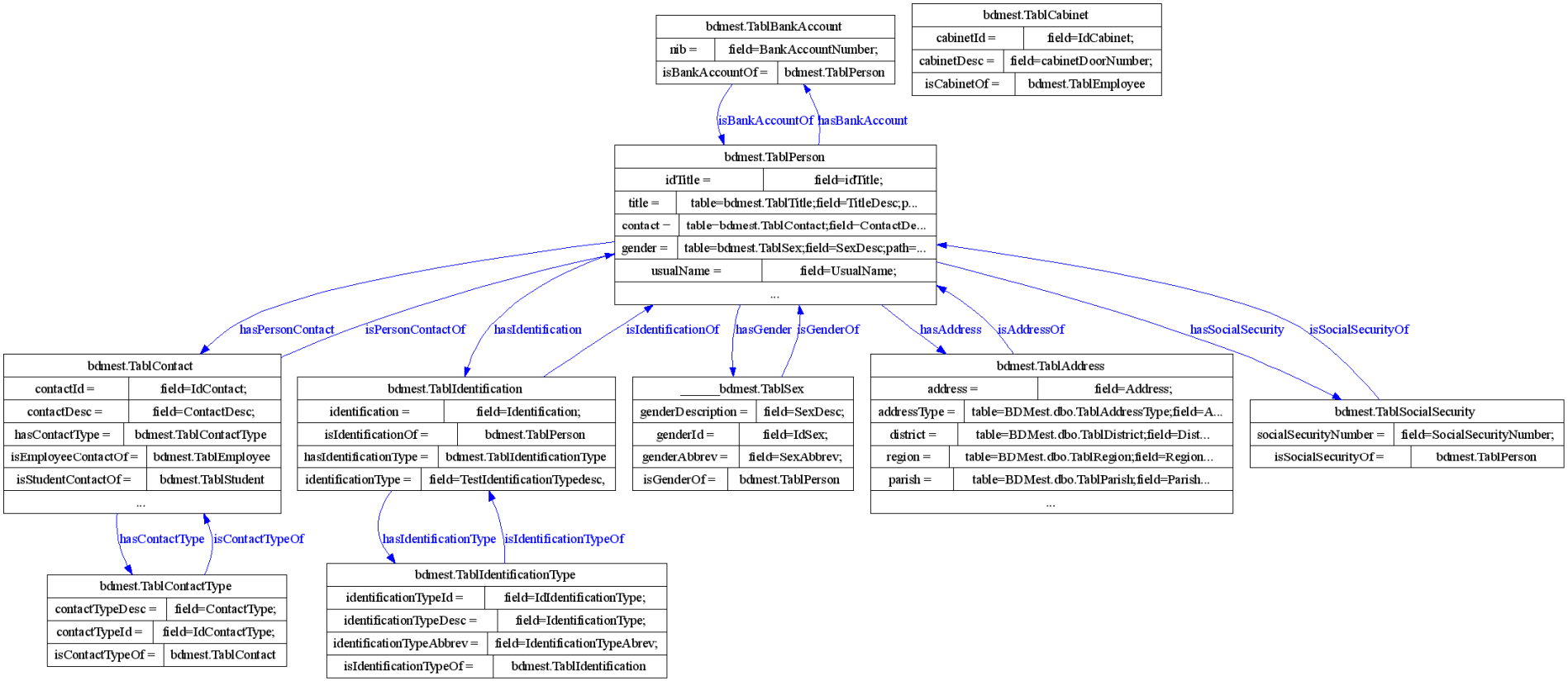


Figure 47 - Ontology instances (example)

The depicted instances contain the mappings from the ontology classes to the database tables.

4.5. INVOKING SWOAT SERVICES – GUI APPLICATION EXAMPLE

This section shows the service request and response of three data services: gender; Person and Identification. The first data service invocation intends to fill a combo box. The second data service which returns data related with personal information, allows getting the name and birth date of a person. The third intends to show all identification (Tax number, ADSE, etc) of a person.

In this case, the GUI application illustrated in Figure 48, is implemented in Windev 9, a development IDE using its demonstration demo. Thus, the chosen development IDE was completely indifferent because nowadays most of the development IDE's support Web Service invocation.

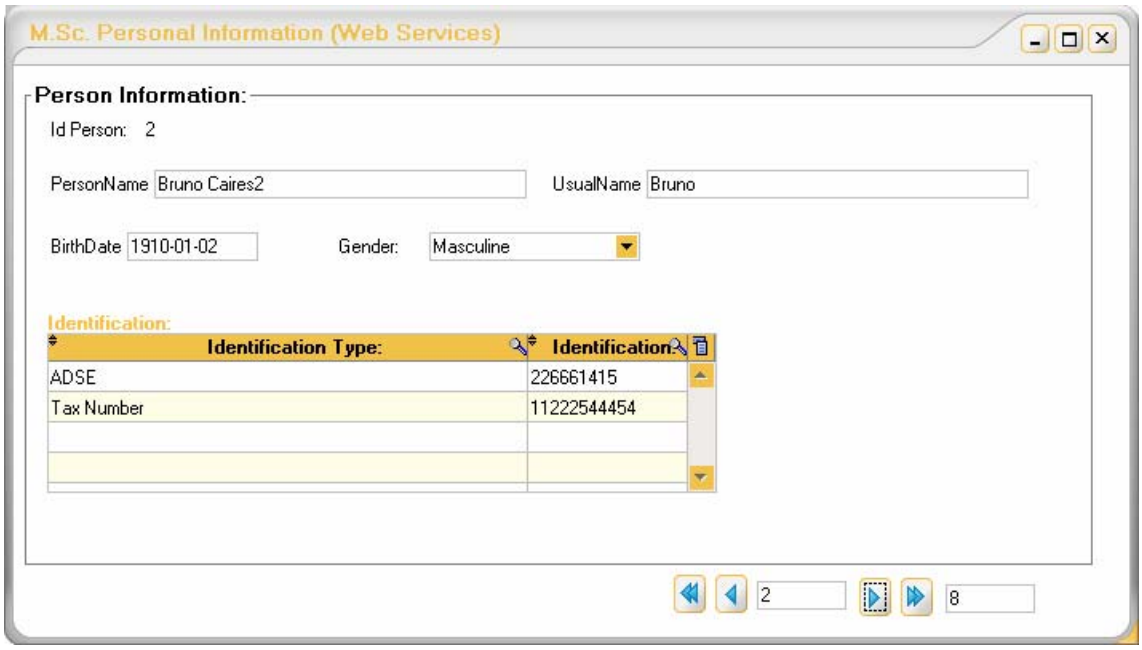


Figure 48 - Application example

The following sections describes in detail each of the three services invoke.

4.5.1.Gender

The following SOAP envelope allows getting all the gender types existing in the database.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <SOAP-ENV:Body>
                <m:genericService xmlns:m="urn:msc" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
                        <String_1 xsi:type="xsd:string">
                                <get_request version="1.0">
                                        <outputFields>
                                                <outputField
name="genderDescription" class="Gender"/>
                                                <outputField name="genderAbbrev"
class="Gender"/>
                                                <outputField name="genderId"
class="Gender"/>
                                        </outputFields>
                                </get_request>
                        </String_1>
                </m:genericService>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Based on this request Swoat builds SQL query in order to return the data. In this case, the generated SQL is:

```
SELECT
        bdmest.TablSex.SexDesc AS genderDescription,
        bdmest.TablSex.SexAbbrev AS genderAbbrev,
        bdmest.TablSex.IdSex AS genderId
FROM
        bdmest.TablSex
```

The service response is the following, which is a transformed XML result of the SQL statement:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <Gender genderId="1" genderDescription="Masculine" genderAbbrev="M"/>
        <Gender genderId="2" genderDescription="Feminine" genderAbbrev="F"/>
</root>
```

The application then fills the combo box with this information, extracting data from the above response using XML XPATH query expressions.

4.5.2.Person

The following SOAP envelope allows getting the name, birth date of a person with idPerson equal to 2, like specified in the xml element filter.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<m:genericService xmlns:m="urn:msc" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<String_1 xsi:type="xsd:string">
        <get_request version="1.0">
                <outputFields>
                        <outputField name="idPerson" class="Person"/>
                        <outputField name="name" class="Person"/>
                        <outputField name="usualName" class="Person"/>
                        <outputField name="idGender" class="Person"/>
                        <outputField name="birthDate" class="Person"/>
                </outputFields>
                <filters>
                        <filter type="=" name="idPerson">2</filter>
                </filters>
        </get_request>
</String_1>
</m:genericService>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SQL statement that returns the requested data:

```
SELECT * FROM
        (SELECT
                bdmest.TablPerson.IdPerson AS idPerson,
                bdmest.TablPerson.PersonName AS name,
                bdmest.TablPerson.UsualName AS usualName,
                bdmest.TablPerson.IdSex AS idGender,
                bdmest.TablPerson.BirthDate AS birthDate
                FROM
                        bdmest.TablPerson
        )innerView
                WHERE (
```

```
                    (idPerson = 2)
                )
```

The SQL statement is then executed and the results transformed reflecting the ontology structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<Person birthDate="1910-01-02 00:00:00.0" usualName="Bruno" idPerson="2"
name="Bruno Caires2" idGender="1"/>
</root>
```

4.5.3.Identification

The following SOAP envelope allows getting all the identifications of a person with idPerson equal to 2.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<m:genericService xmlns:m="urn:msc" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <String_1 xsi:type="xsd:string">
                <get_request version="1.0">
                        <outputFields>
                          <outputField name="idPerson" class="Person"/>
                          <outputField name="identification" class="Identification"/>
                          <outputField name="identificationTypeDesc"
class="IdentificationType"/>
                        </outputFields>
                        <outputProperties>
                                <outputProperty name="hasIdentification"/>
                                <outputProperty name="hasIdentificationType"/>
                        </outputProperties>
                        <filters>
                                <filter type="=" name="idPerson">2</filter>
                        </filters>
                </get_request>
        </String_1>
</m:genericService>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SQL statement generated in order to get the data:

```
SELECT * FROM
        (SELECT
                bdmest.TablPerson.IdPerson AS idPerson,
                bdmest.TablIdentification.Identification AS identification,
                bdmest.TablIdentificationType.IdentificationType AS
identificationTypeDesc
        FROM
                bdmest.TablPerson
```

```
        LEFT JOIN bdmest.TablIdentification
                ON bdmest.TablPerson.IdPerson = bdmest.TablIdentification.IdPerson
        LEFT JOIN bdmest.TablIdentificationType
                ON bdmest.TablIdentification.IdIdentificationType =
bdmest.TablIdentificationType.IdIdentificationType
        )innerView
        WHERE (
                (idPerson = 2)
                )
```

The SQL statement is then executed and the results transformed reflecting the ontology structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<Person idPerson="2">
        <hasIdentification>
                <Identification identification="226661415">
                  <hasIdentificationType>
                        <IdentificationType identificationTypeDesc="ADSE"/>
                 </hasIdentificationType>
                </Identification>
                <Identification identification="11222544454">
                 <hasIdentificationType>
                        <IdentificationType identificationTypeDesc="Tax Number"/>
                 </hasIdentificationType>
                </Identification>
        </hasIdentification>
</Person>
</root>
```

4.6. CONCLUSION

This chapter described a practical application using Swoat. It focussed on the programmer point of view (interface), therefore not showing Swoat intrinsic tasks like the SPARQL query generated to get the mappings from the ontology instances.

The ontology the represents the global information model was first presented. It was also described the existing database schema. The ontology instances that contain the mappings from the ontology to the database were depicted. A GUI application has been presented, illustrating the invocation of three simple services.

5. RELATED WORK

> "The important thing is not to stop questioning. Curiosity has its own reason for existing. One cannot help but be in awe when he contemplates the mysteries of eternity, of life, of the marvellous structure of reality. It is enough if one tries merely to comprehend a little of this mystery every day. Never lose a holy curiosity."
> — Albert Einstein (1879-1955)

This chapter presents the several types of approaches that can be used in order to provide a global virtual view over heterogeneous and distributed sources. The fulfilment to provide a global virtual view over heterogeneous and distributed sources can be addressed by gathering data from the appropriate sources on demand. This approach is named *Mediated approach* and is appropriate for information that changes rapidly and for queries that operate over a vast number of data sources. All the presented approaches are mediated approaches.

There exist several mediated approaches, all of them providing a global view over a set of systems, allowing integration: 'Corporate Ontology Grid' (COG), the 'Mediator envirOnment for Multiple Information Systems' (momis), OBSERVER, the 'Knowledge Reuse and Fusion/Transformation' (kraft), InfoSleuth, 'Ontology-Driven Service-Oriented Integration' (ODSOI) and 'Integration Broker for Heterogeneous Information Sources' (ibhis).

The solutions presented in this chapter will be categorized by each system main differentiating facet. The main categories of the presented solutions are:

- One-to-one mapping: each ontology has to be mapped with other ontology or source or information in a one-to-one relation. The exposed system is OBSERVER.

- Single shared ontology: there exists a central ontology than can be mapped to one or more ontologies or data sources. Presented systems are COG and MOMIS.

- Combined approach: solutions that allow single shared and one-to-one mappings between ontologies and sources of information. The exposed system is InfoSleuth.

- Clustering: Ontology clusters are organized in a hierarchical fashion, where the root node is the most general cluster. A lower level in the hierarchy corresponds to a mode agent specific, less abstract representation of the domain. An example of this approach is KRAFT.

- Service oriented: the ontology provides services, accessible through service invocation. Expounded systems are IBHIS and ODSOI.

The following section will present the most important approaches that provide a global virtual view by using the Mediated approach. Since the presented solutions have not been exhaustively explored in execution, this chapter presents a description of the main functionalities and characteristics of the systems.

5.1. ONE-TO-ONE MAPPING APPROACH

One-to-one mappings are created between pairs of ontologies. This means that each ontology needs to be mapped to other ontologies or to sources of information (like databases) in a one-to-one relation. The system one-to-one system analysed is OBSERVER (observer, 2007), that is a component-based approach to ontology mapping (Mena et al., 1996). It provides brokering capabilities across domain ontologies to enhance distributed ontology querying, thus avoiding the need to have a global schema or collection of concepts.

OBSERVER uses multiple pre-existing ontologies to access heterogeneous, distributed and independently developed data repositories. Each component node has an ontology server that provides definitions for the terms in the ontology and retrieves data underlying the ontology in the component node. An inter-ontology relationship (IRM) provides translation between the terms among the different component ontologies. The IRM contains a one-to-one mapping between any two component nodes. When the user submits a query to the query processor in its own component node, the query processor uses the IRM to translate the query into terms used by other component ontologies and retrieves the results from the ontology servers.

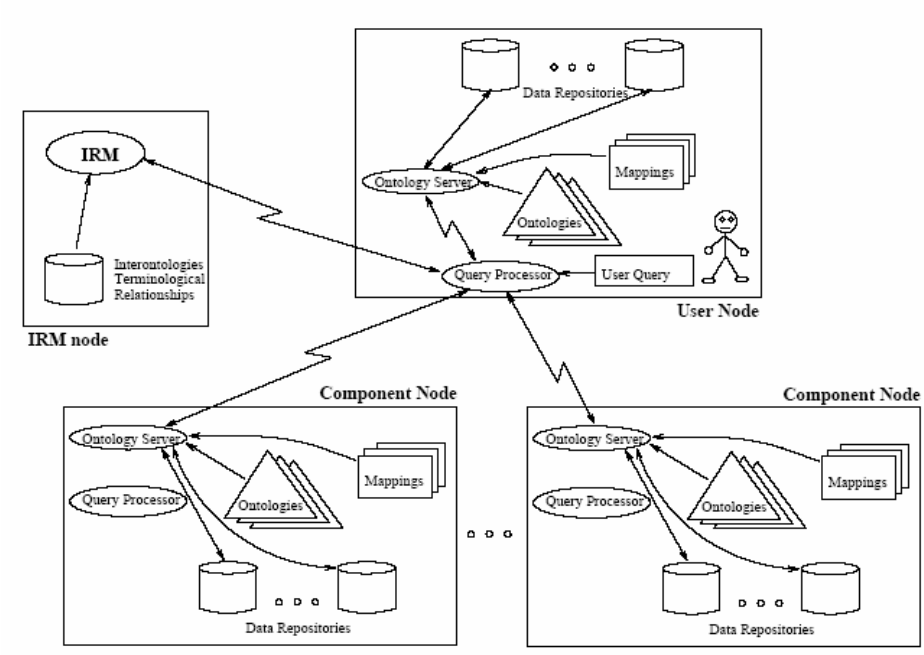


Figure 49 - OBSERVER architecture

As illustrated in Figure 49, the system contains a number of component nodes, one of which is the user node. Each node has an ontology server that provides definitions for the terms in the ontology and retrieves data underlying the ontology in the component node. When the user wants to expand a query over different ontology servers, the original query needs to be translated from the vocabulary of the user ontology into the vocabulary of another ontology (Mena et al., 1996).

Three main OBSERVER characteristics can be stated: (1) Query language in the OBERVER is specific and dependent of the language used to specify the ontology (CLASSIC). (2) In each node, component mapping must exist to all other relevant nodes (one-to-one mapping). (3) The OBERSER is not service oriented.

5.2. SINGLE SHARED ONTOLOGY APPROACH

In a single shared ontology solution, each ontology is mapped to the central ontology. Therefore, the central ontology can be seen as a standard, which the mapped ontologies must respect.

This section described the systems COG and MOMIS.

5.2.1.COG

The COG aims to create a semantic information management in which several heterogeneous data sources are integrated into a global virtual view (Bruijn, 2004b). COG allows the integration of imported RDBMS schema databases, XML Schemas, COBOL copybook and custom wrappers. The workbench that allows the integration is Unicorn Workbench. The tool (workbench) accommodates both the GAV and LAV approach (Bruijn, 2004a).

The user can also issue queries to the mediator that is automatically translated to the respective platforms and schemas of the data sources, where they can be executed. The methodology, consisting of six steps is illustrated in Figure 50.

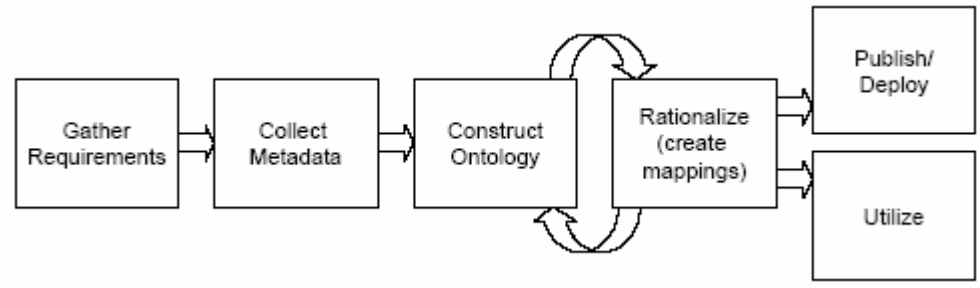


Figure 50 - Semantic Information Management (SIM) methodology

1. Gather Requirements: Is where the requirements for the information architecture are collected and the scope of the project established.

2. Collect Metadata: all data assets relevant to the project are catalogued and the metadata imported.

3. Construct Ontology: means using the imported metadata, creating the ontology through a process of reverse engineering and/or manual identification of classes, properties and business rules in the source schemas.

4. Rationalize: Establish the mappings between the data schemas and the ontology.

5. Publish/Deploy: The ontology, along with the mappings is published to relevant stakeholders.

6. Utilize: Processes need to be created to ensure maintenance of architecture.

The Unicorn Workbench is a java-based tool created by Unicorn, built to support the semantic information management. The architecture of Unicorn Workbench is illustrated in Figure 51. It consists of the information model (ontology) the schemas belonging to the external assets (data sources), the transformations, the queries, and the descriptors (metadata for human readers).

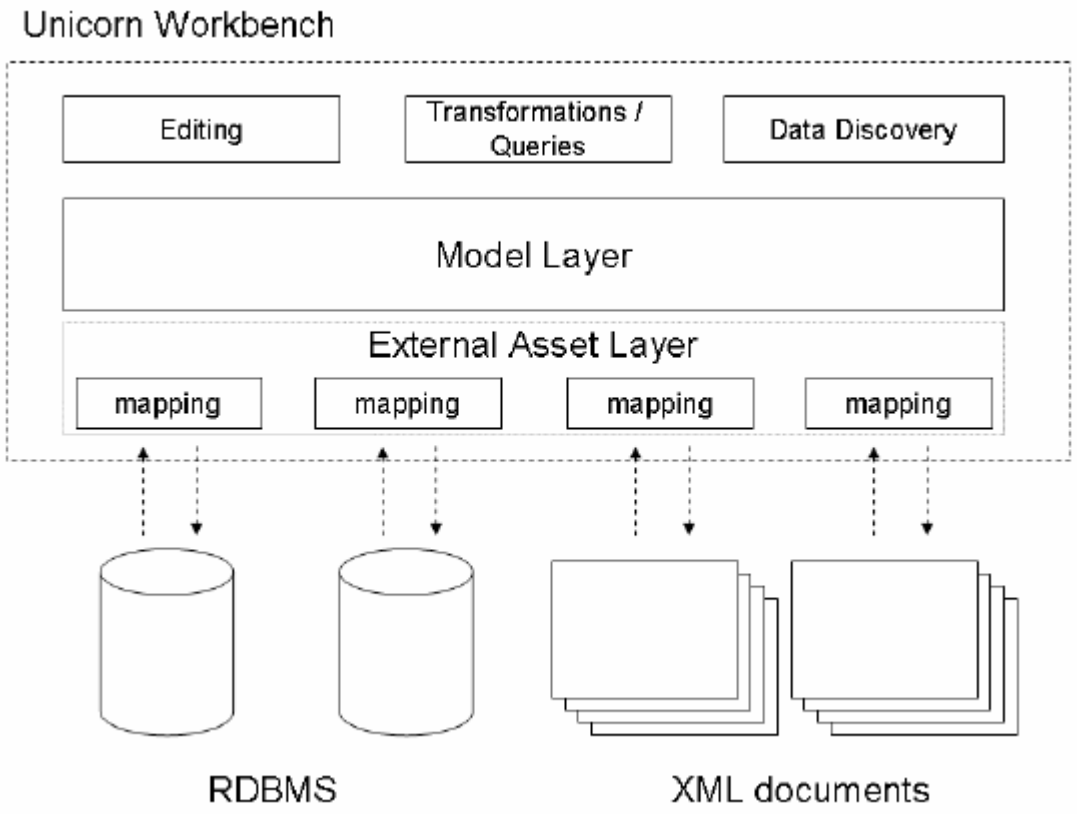


Figure 51 - Unicorn workbench

Ontologies can be created in three ways: reverse-engineering the schema; importing previously created ontologies or by building ontology from scratch. If a data asset has been automatically reverse-engineered, mapping between the ontology and the external asset are automatically built.

Queries in the COG cannot be executed by the Unicorn Workbench and it is not possible to query multiple data sources. Views have to be created in order to permit access to data modelled in the ontology. Queries are similar to SQL and access to the ontology is done via API.

5.2.2.MOMIS

The goal of MOMIS (momis, 2007) is to give the user a global virtual view of the information coming from heterogeneous data sources (Beneventano and Bergamaschi, 2004). MOMIS creates a global mediation schema for the structured and semi structured heterogeneous data sources, in order to provide the user with a uniform query language (Bergamaschi et al., 2004).

The MOMIS framework, as illustrated in Figure 52, consists of a language (ODL-I3) and two main components (Ontology Builder and Query Manager):

- The *ODL-I3* language extends an object-oriented language, with an underlying Description Logic; it is derived from the standard ODL-ODMG.

- The *Ontology Builder*: the various sources integration is performed in a semi-automatic way, by exploiting the knowledge in a Common Thesaurus (defined by the framework) and ODL-I3 descriptions of source schemas with a combination of clustering techniques and Description Logics. This integration process gives rise to a virtual integrated view of the underlying sources (the Global Schema, GVV) for which mapping rules and integrity constraints are specified to handle heterogeneity.

- The *MOMIS Query Manager* is a coordinated set of functions which takes an incoming query, decomposes the query according to the mapping of the GVV on the local data sources relevant to the query, sends the sub-queries to these data sources, collects their answers, performs any residual filtering necessary, and finally delivers the answer to the user.

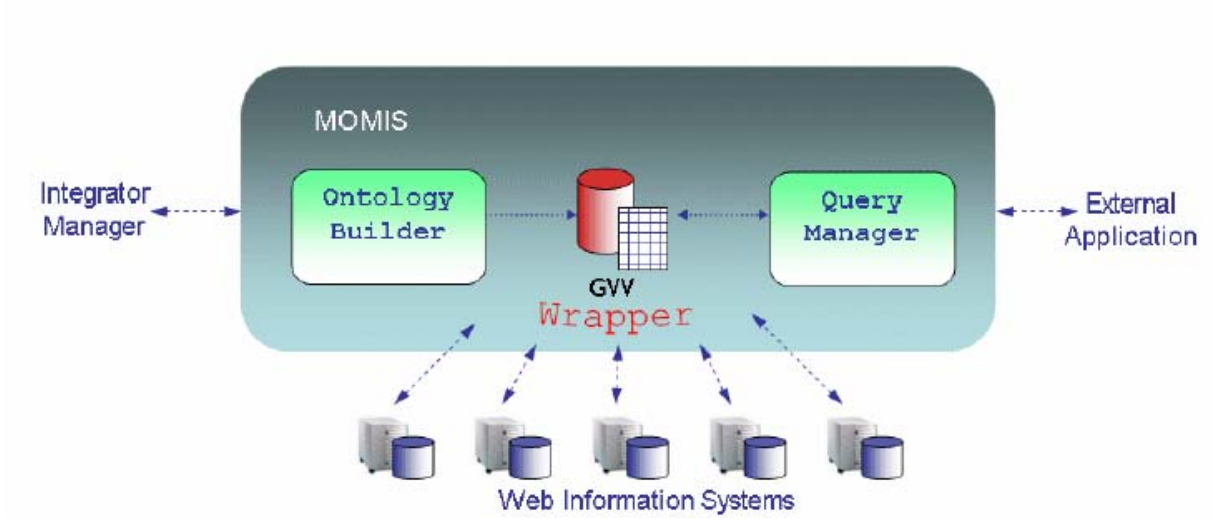


Figure 52 - MOMIS

MOMIS approach is based in GAV approach, which means that the global schema is built, based on the local sources. Other characteristic of MOMIS is that GVV is not expressed in OWL (the W3C language recommended for specifying ontologies). Queries are expressed using a SQL-like language.

5.3. COMBINED APPROACHES: ONE-TO-ONE AND SINGLE SHARED

This type of approach (combined) allow two types of mappings: (1) one-to-one mappings and (2) single shared ontology, therefore allowing the two approaches previously presented. The approach that is analysed in this section is the system named InfoSleuth.

InfoSleuth (InfoSleuth, 2007) is a multi-agent system for semantic interoperability in heterogeneous data sources (Alexiev et al., 2005). It is a deployed, advanced prototype system performing information gathering and analysis over open information sources.

Many applications, including simulation-type applications, are no longer hampered by the availability of data, but rather are concerned with the accessibility of that data (Deschaine et al., 2000). Therefore, there are three separate needs that an information oriented infrastructure can provide to such applications. These are:

- Similar information may exist in many places, but in incompatible forms or formats.
- Applications need to view it as if it were coming from a single source.
- Information processing must integrate both computer-based applications and real machines, such as sensors, giving a uniform methodology to deal with all kinds of information sources and processes.
- Applications often must track the changing state of the information to develop up-to-date information feeds, summaries and analyses.

Agents are used for query and instance transformations between data schemas. An agent is aware of its own ontology and the mapping between the ontology and the data schema, it is aware of the shared ontologies and it can map its ontology to those of other agents. InfoSleuth uses several shared ontologies and individual data sources have mappings to these shared ontologies. The shared ontologies are linked together through one-to-one ontology mappings. The main purpose is to support construction

of complex ontologies from smaller component ontologies so that tools tailored for one component ontology can be used in many application domains.

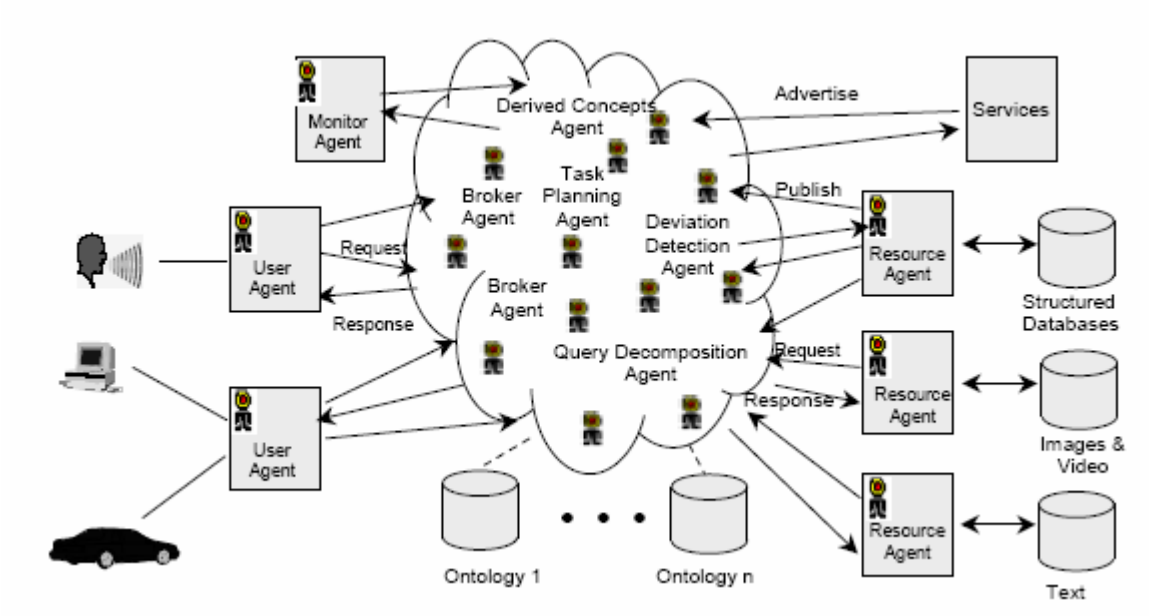


Figure 53 - InfoSleuth architecture

The InfoSleuth consist of different types of agents (Nodine et al., 1999):

- User Agent acts on behalf of the user and maintain the users state. They provide a system interface that enables users to communicate with the systems.
- Resource Agent wraps and activates databases and other repositories of information. They translate queries and data stored in external repositories between their local forms and their InfoSleuth forms.
- Service Agent provides internal information to the operation of the agent system. Service agents:
  - Broker agents collectively maintain the information the agents advertise about themselves.
  - Ontology agents which maintain a knowledge base of the different ontologies used for specifying requests.
  - Monitor agents which monitor the operation of the system.
- *Query and Analysis Agent* fuses and/or analyses information from one or more resources into single results. Query and analysis agents include:

- Multi-resource query agents which process queries that span multiple data sources.

- Deviation Detection agents which monitor streams of data to detect deviations and other data-mining agents.

- *Planning and Temporal Agents* guide the request through some processing which may take place over a period of time. Planning and temporal agents include:

  - Subscription Agents which monitor how a set if information changes over time.

  - Task planning and execution agents which plan the process of user requests in the system.

  - Sentinental Agents: which monitor the information and event stream for complex agents.

- *Value Mapping Agents* provide value mapping among equivalent representation of the same information.

InfoSleuth ontologies are specified in OKBC, which uses object-oriented description logic as an underlying data model. Ontologies are stored in an OKBC server and accessed via ontology agents. These agents provide ontology specification to users for requested information, to resource agents for mapping and to other agents that need to understand and process requests and information in the application domain (Nodine et al., 1999).

5.4. ONTOLOGY CLUSTERING APPROACH

This approach (ontology clustering) is based on the similarity of concepts known to different agents (Visser and Tamma, 1999). Agent systems are designed to operate more effectively in a dynamic information landscape. The communication agents are supposed to have the necessary "intelligence" to be pro-active and to learn about user preferences and environment in which they operate (Alexiev et al., 2005).

KRAFT (kraft, 2007) architecture is designed to support knowledge fusion from distributed heterogeneous databases and knowledge bases. The basic philosophy of KRAFT is to define a "communication space" within certain communication protocols and languages must be respected (Grayy et al., 1997).

The project focuses on the integration of data plus relations between data items (in the form of constraints) rather than merely data or data enriched by context information (Visser and Tamma, 1999).

The KRAFT architecture, as illustrated in Figure 54, is an agent middleware that purposes a set of techniques to map ontologies:

- Class mappings - map a source ontology class name to a target ontology class name.
- Attribute mapping - maps a set of values of a source ontology attribute to a set of values of a target ontology attribute or maps a source ontology attribute name to a target ontology attribute name.
- Relation mapping - maps a source ontology relation name to a target ontology relation name.
- Compound mapping - maps compounds source ontology expressions to compound target ontology expressions.

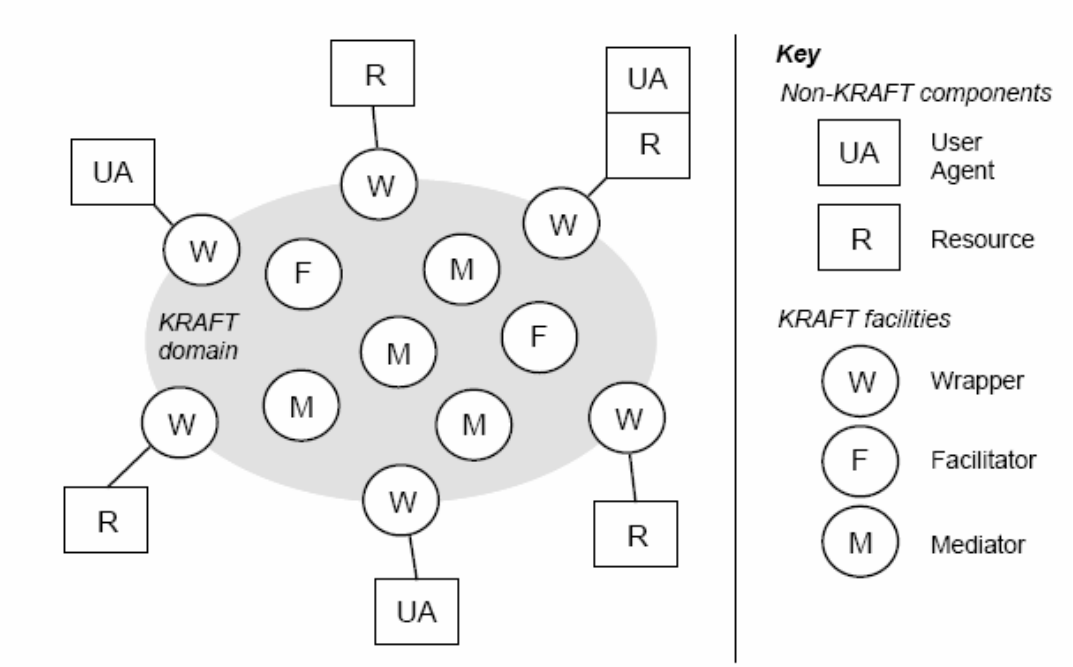


Figure 54 - KRAFT architecture

The KRAFT architecture has three types of agents:

- Wrappers translate the heterogeneous network protocols, schemas and ontologies into the KRAFT application internal standards. A wrapper agent effectively contains a one-to-one mapping between the source schema and the internal ontology.
- Facilitators look up services requested by other agents.
- Mediators provide querying, reasoning and information gathering from the available resources. Mediators contain the mappings between the different ontologies present at the wrappers and they perform the translation between them.

Originally the KRAFT project uses a single-shared ontology in order to enable integration of local ontologies in the overall architecture. Later on, it was suggested the use of ontology clustering. The advantage of ontology clustering is the distinction of more refined and more abstract ontologies, enabling the organization of the ontology into a hierarchical structure (Alexiev et al., 2005).

5.5. SERVICE ORIENTED

Solutions following the service oriented approach intend to be accessible through service invocation. The main purpose is to use open standards to enable service invocation.

Two systems will be analysed: IBHIS and ODSOI.

5.5.1.IBHIS

IBHIS (ibhis, 2007) aims to provide an integrated broker that enables coherent use of a set of distributed, heterogeneous data sources (Kotsiopoulos et al., 2003).

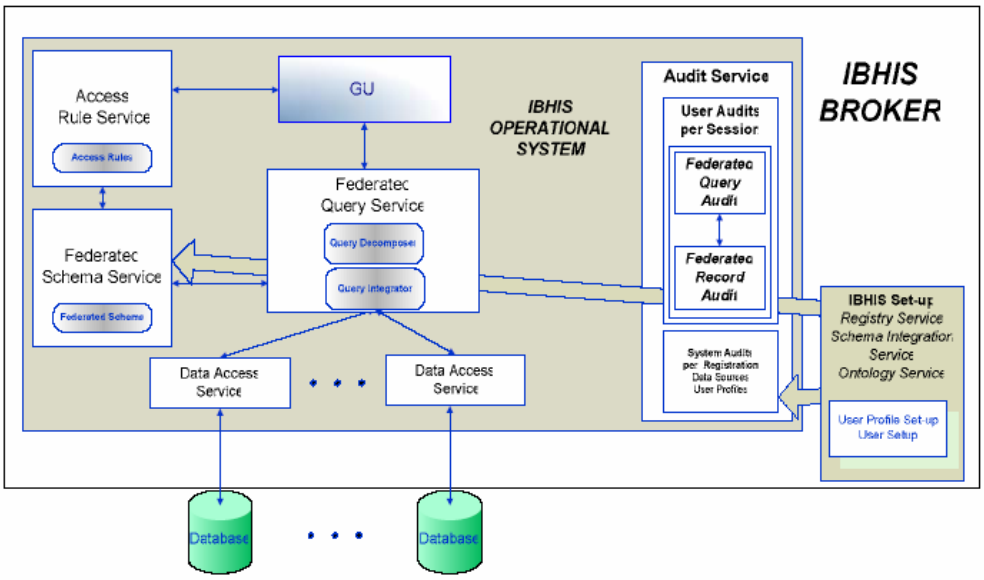


Figure 55 - IBHIS

IBHIS architecture is based on Data as a Service (Zhub et al., 2004), in which through the use of Web Services and open standards and protocols such as java, SOAP, WSDL and UDDI, the database data is accessible as services.

IBHIS users are provided with transparent access to the heterogeneous, distributed data sources once set-up is complete. During set-up, the registration of the users and the underlying data sources takes place. The system administrator constructs the federated schema (global and integrated schema) and resolves all the semantic

differences. Data recorded or created during the system set-up are passed using XML to the Operational System.

The role of the operational system is to receive a query from the user, identify his/her access rights, locate and query the appropriate data sources and return the results to the user.

The Graphical User Interface (GUI) provides the following functions:

- Present the initial login screen to the user.
- Contact the Access Rules Service to authenticate user
- Present a list of available queries to the user, according to their profile.
- Formulate a federated query to pass to the Federated Query Service.
- Display final result

The Access Rule Service (ARS) is responsible for the initial user authentication and subsequent authorisation.

The Federated Schema Service (FSS) maintains the Federated Schema and all the mappings between the Export Schema and the Federated Schema. In future versions the FSS will be consulted by the Federated Query Service during query decomposition.

The Federated Query Service (FQS) contains two sub-modules:

- The Query Decomposer decomposes the Federated Query into a set of local queries; this is done in consultation with the FSS which holds the mappings of the Federated Schema to the Export Schemas.
- The Query Integrator receives the Set of Local Results from the Data Access Service and it integrates them into a Federated Record.

The FQS sends the Federated Query and the Federated Record to the Audit Service. The Audit Service (AS) contains two sub-modules which keep track of every action of IBHIS that needs to be recreated or audited in the future:

- User Audit (per session): holds information such as: user log-in date, time, IP, logout, sequence of Federated Queries, sequence of Federated Record, sequence of sessions, etc.

- System Audit (per Registration): holds information about Data Source (e.g. registration date and time, intervals of availability, etc) and User Setup (e.g. time stamped creation, deletion, profile update, user registration/deletion etc).

One "less positive" issue of IBHIS is that users are not allowed to issue queries dynamically but instead, to choose a pre-configured one.

5.5.2.ODSOI

OSDOI relies on the use of both ontologies and WS technologies (Izza et al., 2005). ODSOI addresses the heterogeneity problem by providing an ontology-driven integration solution that is based on ontology mediation. It is also a service-oriented approach once it uses WS for integrating Enterprise Information Systems, which are defined as an enterprise application system or an enterprise data source that provides the information infrastructure for an enterprise.

ODSOI framework provides a unified framework in order to integrate EIS. Three main types of services are defined: data-services, functional-services and business-services.

Data-Services (DS) are services that expose data sources (EDS – Enterprise Data Sources) as services. *Functional-Services* (FS) are services that expose application systems, fundamentally functional systems (EAS – Enterprise Application Systems) as services. *Business-Services* (observer) are defined as the combination of the above services in order to expose business processes (EPS – Enterprise Process Systems) as services.

ODSOI define three major types of ontologies: information or data-based ontologies, behaviour or functional–based ontologies and process or business-based ontologies.

Data-based ontologies are the most basic ones. They provide semantic description of the data (DS). Theses ontologies are required in all cases, no matter if we leverage functional-based or business-based ontologies. This means that, data integration is a precondition for any other integration type.

Functional-based ontologies define semantic description around functions that provided by the multiple EIS (FS) and that can be remotely invoked. These ontologies are generally required in order to provide a better reuse of functionalities.

Business-based ontologies define semantic description around coordinating business processes. These ontologies are generally required in order to integrate business processes.

The query language is like SPARQL and users can enter their requests using a Java GUI. Then this GUI builds the user request accordingly to SPARQL specification. The user only has access to preconfigured queries, but the main purpose of this approach is to reach a full dynamic version.

5.6. CONCLUSION

Several mediated approaches to provide an integrated view over a set of data sources have been presented. Concerning the solutions, systems were grouped in five main categories: one-to-one mapping; single shared ontology; combined approaches; clustering; and service oriented.

OBSERVER, a one-to-one mappings approach, uses one-to-one mapping which means that the number of connections can be extremely high caused by the one-to-one mappings between pairs of ontologies. OBSERVER uses a proprietary query language that is also dependent of the languages used to specify the ontology. Also, clients are not able to interact with OBSERVER by invoking services over http but instead use proprietary communication protocols.

Regarding to single shared ontology, COG and MOMIS intend to provide only one ontology (and not several as OBSERVER), which reduces the number of connections of one-to-one solutions. The drawbacks of using a single-shared ontology are similar to those of using any standard (Visser and Cui, 1998). For example, it is hard to reach a consensus on a standard shared by many people who use different terminologies for the same domain. Both solutions (COG and MOMIS) are not able to provide services. Also, communication using the http protocol is not allowed.

The combined approach, named InfoSleuth, allows both the previous approaches – one-to-one mapping and shared ontology. This system allows the construction of ontologies from smaller component ontologies, which can be already defined ontologies (that can be reused).

Clustering approaches like the presented system named KRAFT allow that ontology clusters be organized in a hierarchical fashion.

Other solutions allow interaction by invoking services, therefore based on open standards. Examples are IBHIS and ODSOI. While IBHIS allows only issuing preconfigured queries, ODSOI ontologies are used to formally describe the services requests and responses. IBHIS clients are only able to issue preconfigured queries and ODSOI query language is SPARQL thus dependent of the technology used to specify the ontology.

Due the analysis that was done to all systems, the conclusion is that most of the solutions use proprietary standards in order to allow client interaction. From all systems analysed, only the IBHIS and ODSOI allow interaction through services, using the http protocol over the public internet. Thus, these two systems, using Web services, allow interoperability with several types of clients. Also, when referring to the language used to specify the ontology (the global view), all solutions recur to formal specifications, but only the ODSOI solution adopts the OWL, which is a W3C recommendation but as the main drawback that is not being able to generate dynamically SQL queries.

6. CONCLUSIONS

> "A conclusion is the place where you got tired of thinking."
> — Harold Fricklestein

Developed software applications are usually composed by two main components (tiers): database and "front-end". The database stores the data that is manipulated and presented to the user through the "front-end" application, therefore allowing user interaction. This scenario is widely used either in Web based or in GUI applications. Also usual is the fact that one "front-end" application often needs to access to one or more databases, allowing this way integration over several databases.

This scenario presents the following main disadvantages:

- "Front-end" tiers are vulnerable to changes on database structure. This means that if a database is connected to one of more font-end applications and if changes occur in the database, all the applications may have to be changed. Also, "front-end" requests focus not only in what data is intended from the database (that can be from one or more) but also on how to obtain the data which means that this issue can increases the development time of the application.

- The "front-end" application connected with the database depends on the database technology. This means that if the database paradigm changes (ex. from relational to object oriented) or if the database technology changes

(example, from Oracle to MySQL) it is highly probable that all connected clients also have to be changed.

- The functions developed for a specific application (usually centered in obtaining and manipulating data) are usually developed and used only by the application. It is frequent that two tier systems function reusability occurs only in the developed application. Thus, developed functions reusability with other heterogeneous systems is unlikely to be allowable.

The three presented issues present the following challenges:

- How to improve "front-end" developer's productivity by focussing on 'what' data is needed to the application and not with 'how' related issues?
- How to develop "front-end" tiers in order to avoid that changes that occur in the database (syntactic or structural), also imply changes in the application?
- How to hide database vocabulary, providing to stakeholders information models and not complex database schemas?
- How to make the application functions accessible and reusable by other heterogeneous systems, either inside or outside of the organization?

In order to provide a solution to the three mentioned problems, the proposed solution is to develop software systems in three tiers, adding an extra layer between the database and the "front-end" client tiers. Therefore, a framework in order to be deployed between "front-end" clients and databases was developed. The developed solution named Swoat (Semantic and Service Oriented Architecture) framework, takes advantage of emergent technologies in order to provide a possible solution to the above mentioned problems. Before presenting the answers to the above mentioned question, Swoat is going to be briefly presented referring to the used technologies.

Referring to technologies used in Swoat, one key component is ontology that is one of the components of the Semantic Web stack. Ontology is a specification of a conceptualization and therefore a possible solution to be used in Swoat in order to formally describe what information is accessible by clients ("front-end" tier), hiding this way the databases local vocabulary. The implementation of the information model of the organization, also referred as ontology, (recurring to the OWL W3C standard) essentially describes the important domain concepts and the relation between them. It is this implemented information model, recurring to OWL, that can be shared or reused, that contains a direct relation with databases. This relation with

databases exists because the information model describes concepts that usually contain its data stored in a specific databases table. Therefore, mappings from the ontology to the database are required in order to be able to access the databases from the ontology.

In order to allow that the ontology remains independent from specific organization issues, Swoat stores the mappings in the ontology instances. This means that already implemented ontologies can be used simply by creating instances that store the mappings to the databases. When detaching the mappings from the ontology, the information model (ontology) is obtained, independent from specific organization issues. This way, the ontology does not contain any specific implementation details.

Other technology used in Swoat is Service Oriented Architecture (SOA). This way Swoat is accessible to "front-end" clients by invoking services. Recurring to Web Services (WS), Swoat is able to provide services over the public internet and therefore accessible either by internal or external applications. Also, Swoat can be accessible by systems developed in other several implementation languages, allowing this way interoperability.

In order to make "front-end" clients service invocation independent from Swoat implementation technologies, it was developed a Neutral Client Query Language (NCQL). This language is centered in the specification of what information is required to be returned and makes transparent issues related with 'how' (like database location and data – table and field - location). The above mentioned characteristics allow that Swoat avoids the propagation of database changes, wither syntactic or structural, to clients.

Swoat framework is accessible thought services. Being implemented recurring to Web Services (using open standards) allows that "front-end" clients, independently of the implementation language, reuse already implemented functions. Associated with the above mentioned Swoat issues, the NCQL allows that clients focus on what data is needed and not with how to obtain the data from the entire set of databases.

Summing up, and providing answers to the above mentioned questions, the use of ontologies (Semantic Web Technologies) and middleware *avoids the propagation of changes* occurred in the database to all connected clients. For example, if a table is deleted from the database, changes are not required on all "front-end" clients. This is due the fact that the *clients don't have knowledge of the database structure but instead of the information model*. The only relation from the information model to the database is through the mappings. This is the solution that avoids that changes that occur in the

database also imply changes in the "front-end" tiers and also that clients remain unaware of database schemas, therefore hiding the database vocabulary.

Using NCQL, combined with ontologies, means that clients that intend to invoke Swoat services *improves developers productivity* because they only need to specify what data is needed and not how to obtain the data.

Using Service Oriented Architectures, implemented with Web Services, allows that Swoat functions developed in order to be *reused* in a specific application can be reused by other heterogeneous systems, either inside or outside of the organization.

6.1. FUTURE WORK

Swoat has several functionalities to be implemented:

- Implement services to record data in the database. Only the services that allow returning data from the database are actually implemented.
- NCQL needs to be complemented with the capability of using database functions like counting records, summing, etc. Actually this functionality is only allowed by returning all records and then executing them in Swoat middleware.
- Swoat allows manual mapping from the information model (implemented in OWL) to the database tables. It should be created an interface allowing the graphical mappings.
- One powerful ontology mechanism of OWL ontologies is that it is possible the use of synonyms and different languages (Portuguese/English, etc). This means that the user (though NCQL) should be able to make requests using several languages and also using synonyms in the concepts name.
- Allow to define the data type of the ontology classes. Actually, the ontology data types are defined using strings. For example, it is expected that the age of a person is a numeric value. Actually, if the database returns string (ex. 20 years) this value is not validated in Swoat.

6.2. FINAL CONSIDERATION

Due to the running example that was performed, Swoat has shown to be a good solution to quickly create an abstraction layer, deployed between clients and over existing databases. The information model, stored and centralized in Swoat, provides a global and virtual view over the entire set of databases of the organization.

Other Swoat characteristic is the fact that it is a good solution in the development of applications with one database and one "front-end" client. Using Swoat, the integration of the application with other systems is allowed by sharing the application functions by using open standards (Web Services).

Swoat, fully implemented using open source technologies, is indented to be published as an open source product, and due to this fact it is expected this fact helps Swoat improvement and accelerates the implementation of new features.

INDEX

REFERENCES

ALEXIEV, V., BREU, M., BRUIJN, J. D., FENSEL, D., LARA, R. & LAUSEN, H. (2005) *Information Integration with Ontologies*, John Wiley &Sons, Ltd.

ANTONIOU, G. & HARMELEN, F. V. (2004) *A Semantic Web Primer*, MIT Press, 61-149.

BAIRD, S. (Ed.) (2002) *Teach Yourself Extreme Programming in 24 Hours,* Sams.

BENEVENTANO, D. & BERGAMASCHI, S. (2004) T*he MOMIS Methodology for Integrating Heterogeneous Data Sources*, http://dbgroup.unimo.it/prototipo/paper/ifip2004.pdf.

BERGAMASCHI, S., BENEVENTANO, D., CORNI, A., GELATI, G., GUERRA, F., MISELLI, D., MONTANARI, D. & VINCINI, M. (2004) *The MOMIS System*.

BERNERS-LEE, T. (2000) *Semantic Web on XML*.

BERNERS-LEE, T., HENDLER, J. & LASSILA, O. (2001) *The Semantic Web. Scientific America.*

BERRY, D. (2005) The user experience, Part 1 - It's a matter of style -- GUI versus the Web. in TEAM, I. E. O. U. (Ed.), http://www.ibm.com/developerworks/web/library//w-berry2.html, accessed on 2006-07-03.

BRUIJN, J. D. (2004a) *Best Practices in Semantic Information Integration*.

BRUIJN, J. D. (2004b) *Semantic Integration of Disparate Sources in the COG Project*, http://www.inf.unibz.it/~jdebruijn/publications/COG-ICEIS2004.pdf.

BUCHMANN, A., COULSON, G. & PARLAVANTZAS, N. (2006) *Middleware*, IEEE Distributed Systems.

BURETTA, M. (1997) *Data Replication: Tools and Techniques for Managing Distributed Information,* John Wiley & Sons Inc.

BUSSLER, C. (2003) *B2B Integration,* Springer-Verlag.

CAIRES, B. & CARDOSO, J. (2006) *Using Semantic Web Technologies to Build Adaptable Enterprise Information Systems*, International Journal of Interoperability in Business Information Systems (IBIS), issue 3, 41-58.

CAIRES, B. & CARDOSO, J. (2007) *Using Semantic Web and Service Oriented Technologies to Build Loosely Coupled Systems: SWOAT – A Service and Semantic Web Oriented Architecture Technology*, ICEIS 2007. Funchal.

CARDOSO, J. & SHETH, A. (2006a) *Programming the Semantic Web.* Semantic Web Services, Processes and Applications, Springer.

CARDOSO, J. & SHETH, A. (2006b) *Semantic Web Services, Processes and Applications*, Springer.

CHRISTENSEN, E., CURBERA, F., MEREDITH, G. & WEERAWARANA, S. (2001) *Web Services Description Language (WSDL) 1.1.*

CYCL (2007) http://www.cyc.com/cycdoc/ref/cycl-syntax.html

CYCORP (2006) *Cyc KNowledge Base*, http://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/36.htm.

DAML (2001) DAML+OIL, http://www.daml.org/.

DESCHAINE, L. M., BRICE, R. S. & NODINE, M. H. (2000) *Use of InfoSleuth to Coordinate Information Acquisition, Tracking and Analysis in Complex Applications*, Advanced Simulation Technologies.

DODDS, L. (2005) *Introducing SPARQL: Querying the Semantic Web*, http://www.xml.com/pub/a/2005/11/16/introducing-sparql-querying-semantic-web-tutorial.html, accessed on 21-01-2007.

ERL, T. (2005) *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*, Prentice Hall

FIRESTONE, J. M. (2002) *Enterprise Information Portals and Knowledge Management*, KMCI Press.

FREMANTLE, P., WEERAWARANA, S. & KHALAF, R. (2002) *Enterprise Services*, IN ACM, C. O. T. (Ed.).

GENESERETH, M. (2006) *Knowledge Interchange Format (KIF).*

GRAYY, P. M. D., PREECEY, A., FIDDIANZ, N. J., GRAYZ, W. A., BENCH-CAPON, T. J. M., SHAVE, M. J. R., AZARMI, N., WIEGAND, M., ASHWELLZ, M., BEER, M., CUI, Z., DIAZ, B., EMBURYY, S. M., HUIY, K., JONESZ, A. C., JONES, D. M., KEMPY, G. J. L., LAWSONZ, E. W., LUNN, K., MARTIZ, P., SHAOZ, J. & VISSER, P. R. S. (1997) *KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases*, http://doi.ieeecomputersociety.org/10.1109/DEXA.1997.10004.

GRUBER, T. (1993) *A Translation Approach to Portable Ontology Specifications*, in http://ksl-web.stanford.edu/ksl_abstracts/ksl-92-71.html.

HAASE, P., BROEKSTRA, J., EBERHART, A. & VOLZ, R. (2004) *A Comparison of RDF Query Languages*, Third International Semantic Web Conference.

HENDLER, J., BERNERS-LEE, T. & MILLER, E. (2002) *Integrating Applications on the Semantic Web*, Institute of Electrical Engineers of Japan.

HIBERNATE (2006) *Hibernate Reference Documentation*, http://www.hibernate.org/, accessed on 2006-02-17.

HOHPE, G. & WOOLF, B. (2004) *Enterprise Integration Patterns*, Addison-Wesley.

IBHIS (2007) http://www.informatics.manchester.ac.uk/ibhis/.

INFOSLEUTH (2007) http://www.argreenhouse.com/InfoSleuth/.

IZZA, S., VINCENT, L. & BURLAT, P. (2005) *A Unified Framework for Enterprise Integration An Ontology-Driven Service-Oriented Approach*.

KIF (2007) http://logic.stanford.edu/kif/specification.html

KODALI, R. R. (2005) *What is service-oriented architecture? - An introduction to SOA*, http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html.

KOTSIOPOULOS, I., KEANE, J., TURNER, M., LAYZELL, P. & ZHU, F. (2003) *IBHIS: Integration Broker for Heterogeneous Information Sources*, 27th Annual International Computer Software and Applications Conference (COMPSAC 2003).

KRAFT (2007) http://www.csc.liv.ac.uk/~kraft/.

LANZERINI, M. (2002) *Data Integration: A Theoretical Perspective*, http://www.cs.uwaterloo.ca/~gweddell/cs848/Lenzerini02.pdf.

LASSILA, O. & SWICK, R. (1999) *Resource Description Framework (RDF) model and syntax specification*.

MAHMOUD, Q. H. (2005) *Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)*.

MCGUINNESS, D. L. & HARMELEN, F. V. (2004) *OWL Web Ontology Language Overview*.

MENA, E., KASHYAP, V., SHETH, A. & ILLARRAMENDI, A. (1996) *OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies*

MOMIS (2007) http://www.dbgroup.unimo.it/Momis/.

NODINE, M., FOWLER, J., KSIEZYK, T., PERRY, B., TAYLOR, M. & UNRUH, A. (1999) *Active Information Gathering in InfoSleuth*

NOY, N. F. & MCGUINNESS, D. L. (2001) *Ontology Development 101: A Guide to Creating Your First Ontology*, http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html.

NUNES, N. J. (2001) *Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach*, Departamento de Matemática e Engenharias. Funchal, Universidade da Madeira.

OBSERVER (2007) http://sid.cps.unizar.es/OBSERVER/.

PENNINGTON, C., CARDOSO, J., MILLER, J. A., PATTERSON, R. S. & VASQUEZ, I. (2007) *Introduction to Web Services*.

RDF (2007) http://www.w3.org/RDF

ROSETTANET http://portal.rosettanet.org/cms/sites/RosettaNet/.

RUGGIERO, R. (2005) *Integration Theory*. DM Direct Newsletter.

SILVA, A. & VIDEIRA, C. (2001) *UML Metodologias e Ferramentas CASE*, Centro Atlantico.

SOMMERVILLE, I. (2001) *Software Engineering (6th Edition)*, Addison Wesley.

SPARQL-W3C (2005) *SPARQL Query Language for RDF - W3C Working Draft 20 February 2006*, http://www.w3.org/TR/2006/WD-rdf-sparql-query-20061004/.

TAYLOR, J. (2004) *Enterprise Information Integration: A New Definition*, IN CONSORTIUM, I. (Ed.) Thoughts from the Integration Consortium.

TAYLOR, J. (2006) *Enterprise Information Integration: A new Definition*, Integration Consortium.

VISSER, P. & CUI, Z. (1998) *On accepting heterogeneous ontologies in distributed architectures*, ECAI98 Workshop on Applications of Ontologies and Problem-solving Methods. Brighton.

VISSER, P. & TAMMA, B. (1999) *An experience with ontology clustering for information integration*, IJCAI-99 Workshop on Intelligent Information Integration in Conjunction with the Sixteenth International Join Conference on Artificial Intelligence. Stockholm.

W3C-SCHEMA (2004) *RDF Vocabulary Description Language 1.0: RDF Schema*, IN W3C (Ed.), http://www.w3.org/TR/rdf-schema/, accessed on 2007-01-25.

W3C-SOAP (2003) *SOAP Version 1.2 Part 0: Primer*.

W3C-SW (2006) *Semantic Web*, http://www.w3.org/2001/sw/.

W3C-WS (2006) *Web Services*, http://www.w3.org/2002/ws/.

W3SCHOOLS (2006) *XML Schema Tutorial*, http://www.w3schools.com/schema/default.asp.

WIKIPEDIA *Universal Description Discovery and Integration*.

WSA-W3C (February 2004) *Web Services Architecture*, W3C Working Group Note 11. http://www.w3.org/TR/ws-arch/, accessed on 2006-08-20.

ZHUB, F., TURNERA, M., KOTSIOPOULOSC, I., BENNETTB, K., RUSSELLD, M., BUDGENA, D., BRERETONA, P., KEANEC, J., LAYZELLC, P. & RIGBYD, M. (2004) *Dynamic Data Integration using Web Services*.