Using Semantic Web Technologies to Build Adaptable Enterprise Information Systems

Bruno Caires
Department of Communications and Computing
University of Madeira
Funchal, Portugal
bruno.caires@uma.pt

Jorge Cardoso
Department of Mathematics and Engineering
University of Madeira
Funchal, Portugal
jcardoso@uma.pt

Abstract: In most existing software systems, client applications are tightly coupled to database systems (client/server), which imply that when changes occur in the database, those changes also have to be propagated to all connected clients. Another issue is that since several database engines may exist in the organization, in most cases relational databases, the integration may be a very difficult process. To overcome the above-mentioned problems, we propose a solution based on a middleware located between clients and database servers that provide both an abstraction layer and a unified view over a set of databases. The middleware is based in semantic Web technologies and uses a semantic global model specified in OWL. Interoperability with other systems/organizations is achieved providing the middleware services as Web Services. Therefore, our approach allows clients to be loosely coupled from the database servers, minimizing maintenance when changes occur.

Introduction

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, the PDA calendar synchronize with the corporate calendar server, it seems that any application can be made better by integrating it with other applications [HW04]. To confirm the importance that integration has assumed, studies show that European corporations spend over 10 billion euros in information integration [ABBFLL05]. In addition, integration costs assume an average of 24% of yearly IT budget [Yag02]. Therefore, integration is one of the most important challenges that organizations face today.
Semantic and semantic Web technologies offer a new way to integrate data and applications [O06]. These new technologies have find one of their first commercial users in organizations facing data integration needs [O06] and always seeking for better data integration solutions. According to TopQuadrant, a consulting firm that

specializes in semantic Web technologies, the market for semantic technologies will grow at an annual rate of between 60% and 70% until 2010.
Based on semantic technologies [Pal01], 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 [BM02]. Toward this objective, and in order to achieve the "well defined meaning of information", a fundamental concept in the centre of the semantic Web is ontology. An ontology is a formal, explicit specification of a shared conceptualization [Grub93] allowing the definition of concepts, attributes and relations between concepts. Ontologies allow data to be defined and linked in a way that enables its use for more effective discovery, integration, re-use across various applications and machine processing [BM02].
Our approach for integrating several Relational Database Management Systems (RDBMS) is achieved by using an abstraction layer, providing a virtual view over a set of data sources in order to allow the data accessibility in real time. This virtual view represents the knowledge that the users of the system want to store and access, rather than the data that implements that knowledge. The global model, built using semantic Web technologies is not only human readable but also computer readable. Applications access data sources through a global virtual view, abstracting from aspects like data source type, connection type and data source query language, focussing on the 'what data' and not on 'how to get the data'. Ontologies expressed in Web Ontology Language (OWL) constitute a good candidate to represent the virtual view of our system. In fact, the shared conceptualization (ontology) can be an abstract model for all the enterprise domain concepts. These domain concepts are explicitly defined and related independently of the underlying applications. This model is created independently from the data sources, allowing reuse and distribution of the created ontology among stakeholders. It should describe the most accurate domain model of the organization, not being limited or restricted by any existing application or data source schema.
A middleware system, that implements the global view, should be built improving reuse, evolution and organization of the developed system [Rit05]. Thus, one possible approach to break apart a complicated software system is layering [FRFHM02]. The architecture of our system is based in three layers: *data source*, *domain* and *interface,* described in the following sections. The interoperability of our system is achieved through the use of a Service Oriented Architecture (SOA) [He03] that relies on Web Services [WS] to expose and allow clients (both applications or external organizations) to interoperate with the virtual view.

Data Source Heterogeneity

When several database systems exist in an organization, a common problem associated to the creation of a global view is heterogeneity. It occurs when there is a disagreement about the meaning, interpretation or intended use of the same or related data. Four types of information heterogeneity may arise: system heterogeneity, syntactic heterogeneity, structural or schematic heterogeneity, and semantic heterogeneity [CA06]:

- System heterogeneity: Applications and data may reside in different hardware platforms and operation systems.

- Syntactic heterogeneity: Information resources may use different representation and encodings of data. Syntactic interoperability can be achieved when compatible forms of encoding and access protocols are used to allow information systems to communicate.
- Structural heterogeneity: Different information systems store their data in different data models, data structures and schemas.
- Semantic heterogeneity: The meaning of the data can be expressed in different ways leading to heterogeneity. Semantic heterogeneity considers the content of an information item and its intended meaning.

The use of Web Services can solve the syntactic and system heterogeneity. XML and XSD (schemas) [W3CXC] can solve the structural heterogeneity because a XML file that respects a specific XSD Schema has a well-defined structure. Using OWL, as a shared ontology, semantic heterogeneity is resolved [CA06]. These technologies are the foundation of the system we have developed.

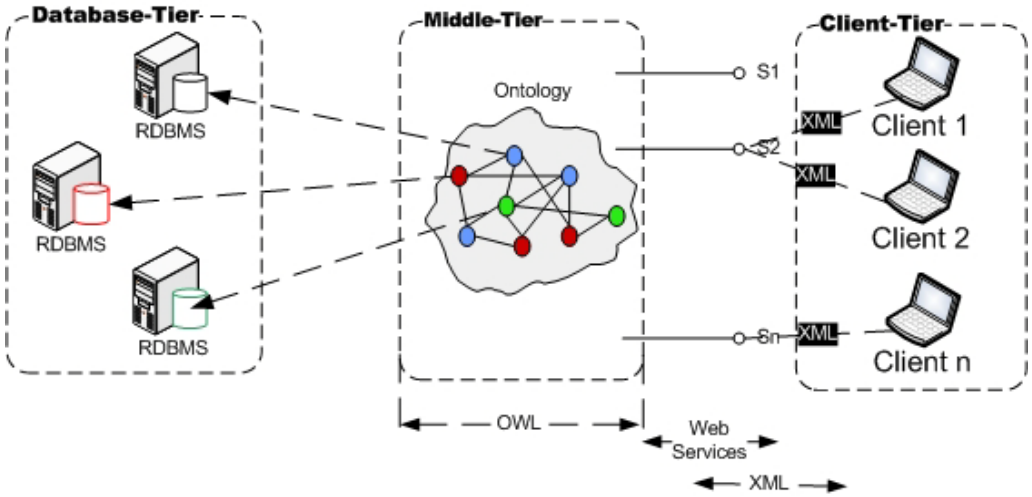

Figure 1: Integration using a shared Ontology

As illustrated in Figure 1, the middleware (middle-tier located between the client-tier and the database-tier) contains the global virtual view over a set of databases. The global virtual view is specified using ontology, described in OWL. There are services (S1, S2, Sn) exposing and allowing access to databases through Web Services. Service requests and responses are XML messages. Therefore, syntactic, system and structural heterogeneity are achieved.

Motivating Scenario

Let us suppose an organization that has several software systems, each one connected to a particular RDBMS database. Examples include the human resource management system, the accounting system, among others. Typically, developed applications followed two tier (client/server) architecture. Client applications were commonly "Commercial of-the-shelf" (COTS), implemented in a language such as java or php, while database servers were engines such as mysql. With this approach, clients were directly connected to servers (databases) and business rules

were stored in the database server or in each application. In this specific scenario, business rules represent the rules that must be followed in order to insert or retrieve data from the database. It may also imply calculations and data transformation. If the business rules are stored in the application, it makes difficult its reutilization. It also implies a new release of the application for each change. If the business rules are stored in the server (database) there is a dependence of the database server technology, which makes difficult its change. Also, in most database systems, the programming language is data-centric, therefore not appropriated for business rules manipulation. Nevertheless, having the development time as an advantage, a disadvantage from the typical client/server architecture is that if changes occur in the database schema, all the clients need to be changed. Changes in the database can be either structural (change completely the structure of the database) or just a change in the name of a table or attribute. Applications frequently suffer changes along their lifetime. These changes are motivated by immediate needs, maintenance tasks and changes in the evolving environment. Consequences of this situation are backward and expensive modernization and adjustment of the built systems.

In our approach, we developed a middle-tier that besides acting as an abstraction layer [HL], is also suitable to the integration of data from multiple systems into a unified, consistent and accurate representation geared toward the viewing and manipulation of data. Through the middle-tier, data is aggregated, restructured, relabelled and presented to the user [T06], therefore centralizing the business rules.

With the use of semantic Web technologies to develop the middleware and by creating a model of data entities (ontology), mapping those entities to their respective sources and exposing its services as Web Services we intend to:

- Isolate changes that may occur in the database. When the database changes, it is not necessary to change all the clients.
- Increase productivity of developers presenting them the domain model and with not complex database schemas.
- Allow the 'interface developers' (like php, asp, etc) to make queries dynamically and in the domain language described by the ontology, abstracting from technical aspects like in which database is the data, type of server and SQL query language.
- Minimize the time developers spend learning our database system and creating access points to the information (views and stored procedures).
- Increase productivity and reduce maintenance to the developed solutions.
- Achieve interoperability with other applications/organizations using Web Services.

The Prototype

In this section, we start by defining the methodology used to build the abstraction layer (middleware) by providing a virtual view over a set of RDBMS data sources. It follows with the architecture of the developed prototype, describing each of its layers. The mapping process to the data sources and the XML query language that allows users (even non-technical) and applications (internal or external to the organization) to make requests to the created virtual view are presented. As an

example, we will use the "personal data" ontology to show a running example. This ontology describes personal data such as name, birth date, address, contact and identification associated to a person.

Methodology

Our approach is based in the Semantic Information Management (SIM) methodology [BJ04]. 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 [ABBFLL05, BJ04 and Bru04].

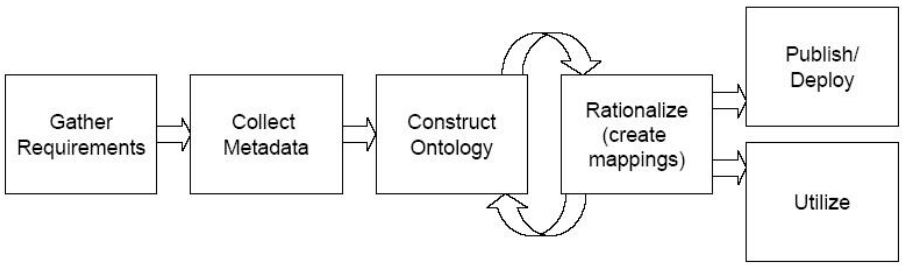

Figure 2: The Semantic Information Management Methodology

As illustrated in Figure 2, 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.

Our SIM based methodology differs from the original definition because the ontology created is not generated by 'reverse engineering' the database schemas but instead generated from scratch (LAV) [LAN02], and then map it to the database object (table or view) that stores the data described by the concept. This way, the created ontology, describes the 'as it should be' and not the 'as is implemented'. Because the ontology is generated from scratch, already created ontologies can be reused, and the created ontology can be distributed. Another aspect that motivated the LAV approach is that the ontology is going to describe the structure of the XML response of a service request. This is going to be illustrated in more detail in the "Querying the middleware" section.
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 advantage that motivated its use is that in most cases, if the database changes (databases externally or internally developed) the ontology remains the same, which imply that clients do not have to be changed.

Architecture

The architecture of our middle-tier is three layered (data source, domain and presentation layer), as depicted in Figure 3. In the next sections, we describe each of the three layers.

Data Source Layer

This layer is responsible for the communication with databases (RDBMS). Our solution uses Hibernate [HIB]. Hibernate design goal is to relieve the developer from 95% of common data persistence related programming tasks by eliminating the need for manual, hand-crafted data processing using SQL and JDBC [HIB]. It is an open-source product developed in java and increases the developer productivity enabling developers to focus more on the business problem. 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, Point base, Progress, Front Base, Ingres, Informix and Firebird [J06].
As outlined in Figure 3, the data source layer contains a set of classes that represent an interface (access point), allowing access to the objects of database servers. This layer is responsible for executing SQL statement in order to get data from the database.

Domain Layer

This layer contains the domain model described in OWL. Domain concepts, relations between them and the rules associated are all presented in the OWL model. The ontology is queried using SPARQL [PS06], which is a W3C ontology query language. Our domain layer also contains the mapping to the data sources, stored in the instances of the ontology, as illustrated in Figure 3. This is going to be described in more detail in the 'Querying the Middleware' section. This layer receives requests in SPARQL and executes the queries in order to extract the mapping data from the instances. Then, using a developed "query generator module" the SQL statement that allows getting the data from the data sources layer is generated using the data stored in the ontology and in the ontology instances. Then, Data is transformed to a XML format. This is going to be illustrated in more detail in the next sections.

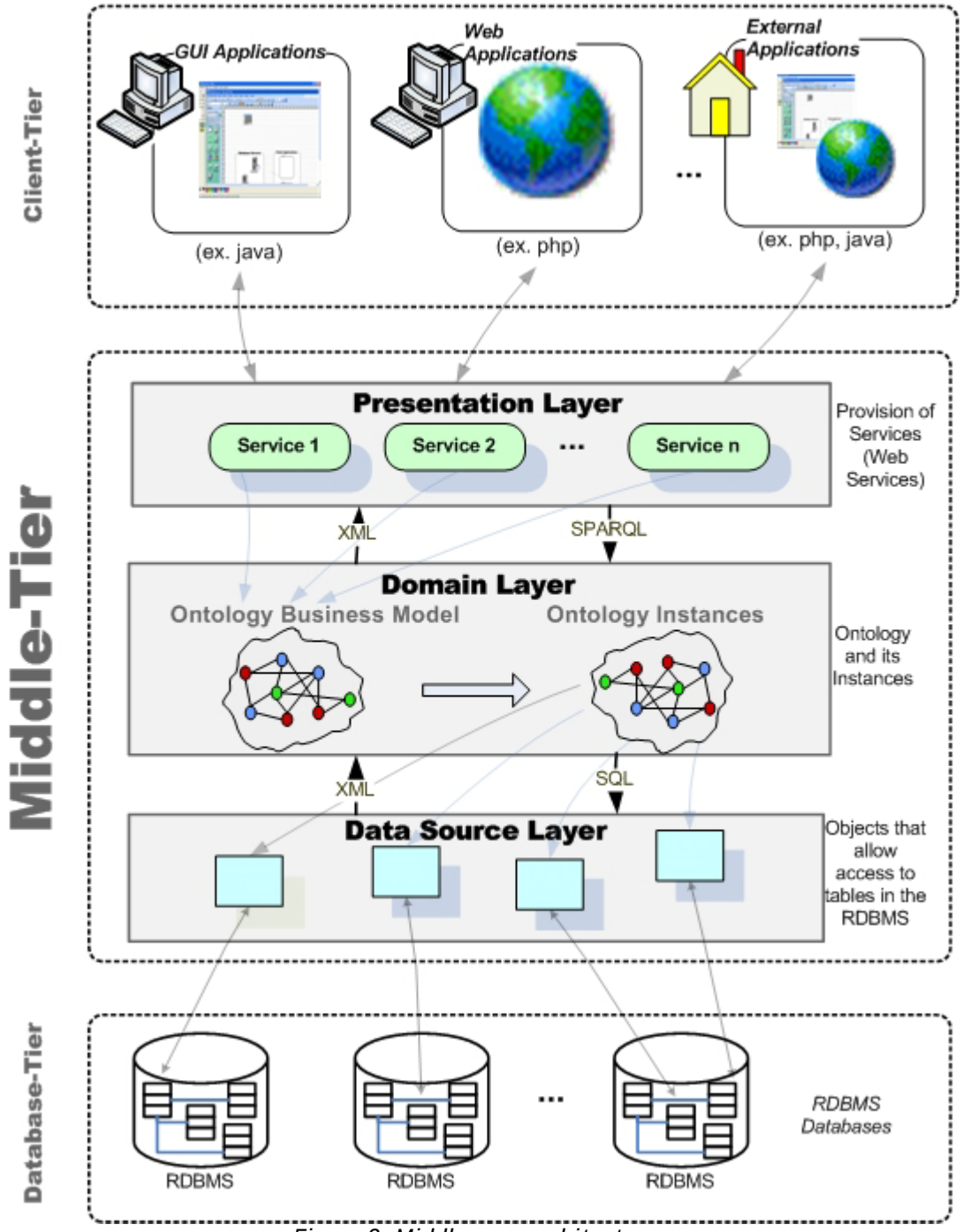

Figure 3: Middleware architecture

Presentation Layer

This layer receives the request message in XML from clients and provides responses in XML. Web Services are used in order to provide the services, which mean that responses are encapsulated in SOAP [W3CSP]. This layer also has the responsibility to transform data that is going to be returned to clients (according to a specific XML format) using XSLT style sheets [W3CXS]. As shown in Figure 3, this layer sends a SPARQL statement to the domain layer. The SPARQL expression is generated by

transforming the client request that is described in detail in the *Query Language* section.

Ontology to Represent Domain and Instances to Store Mappings

In this section, we discuss the ontology that resides in the domain layer. First, important domain classes and respective attributes were described. Examples are illustrated in Figure 4. All attributes should be as string type because the instances of the created classes will contain the mappings to the data sources. This way, each ontology instance will contain the database table name that stores the data, as is going to be described in more detail in the next section. Ontology relations describe the relations that exist between domain concepts. For example, the domain class Person is connected to Address through the property hasAddress, as depicted in Figure 5. 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.

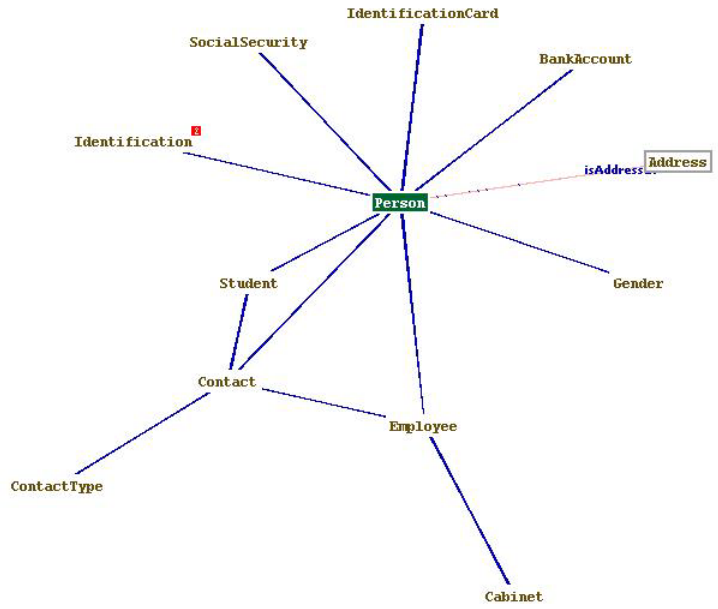

Figure 4: Partial view of the classes of Personal Data ontology (using TGViz)

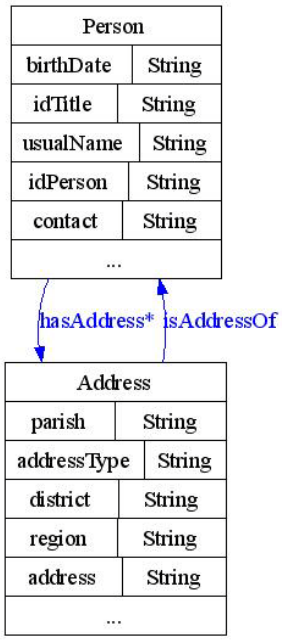

Figure 5: Person and Address domain concepts (using Ontoviz)

Mappings to Data Source Layer

After the creation of the ontology (or reuse of an already defined ontology), instances containing the mappings to the data sources should be created. The mappings will store information in order to allow the construction of an SQL statement that will be executed in the data source layer in order to get the desired information.

In order to generate the correct SQL statement, two types of mappings should be created: domain classes and attributes to database tables and fields, and domain relations to database relations.

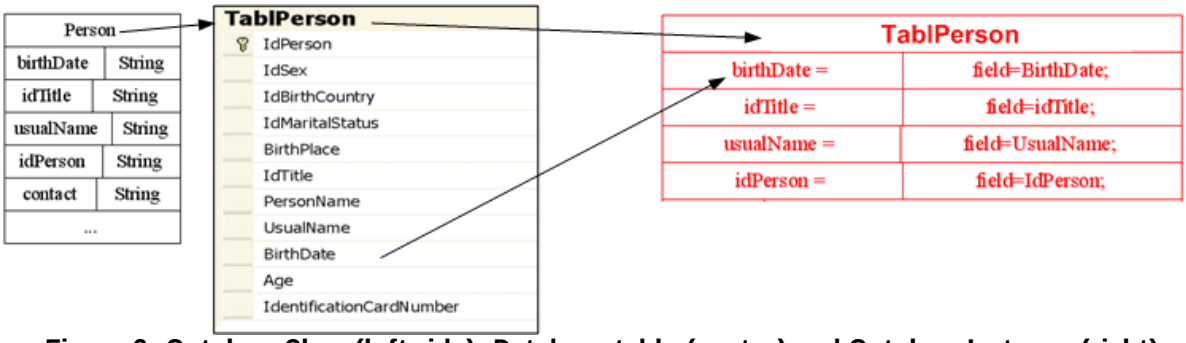

Figure 6: Ontology Class (left side), Database table (centre) and Ontology Instance (right)

Several cases may exist in mapping domain classes to database tables:

- One class of the ontology may contain data from only one database table.
- One class of the ontology may contain data from two or more database tables.
- One table from the database provides data to two or more domain classes.

In all of these cases, the name of the instance should be equal to the name of the object in the data source layer that allows access to the table in the database. The fields that belong to the table are mapped starting by *field=???;* (??? Represents variable text). The fields that do not belong to the instance, should be mapped using the following nomenclature: *table=???;field=???;path={???}*:

- *table* represents the name of the table that contains the specified field.
- *field* represents the name of the field that is stored in the table (previously presented).
- *path* represents the path to reach the table. The content of Path is *field=???;table=???;field=???...field=???;table=???;field=???*;. It should always start and end with field. At least one field attribute is required.

As an example, using a domain class to database table mapping (illustrated in Figure 6), the name of the instance of the class Person should be equal to the respective object in the data source layer (TabPerson in this case). Attributes of the created instances should contain the name of the field in the specified table name, as depicted in Figure 6 by *birthDate*.
While in the domain model (described In OWL) relations connect domain concepts, in the database relations connect tables. In order to map ontology relations to database relations, our solution is to annotate the ontology relations. The content of the annotation property, named *relationAttribute*, should follow the format *field=???table=???field=???...field=???table=????field=???*. It should start and end with 'field='. The number of 'tables=' depends of the number of intermediary tables that exist.
Exemplifying, in Figure 7, the domain classes Person and Address are shown. These two ontology classes will be mapped to the tables TablPerson and TablAddress (depicted in Figure 8), respectively. As illustrated in Figure 8, an intermediary table named TablPersonAddress 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 hasAddress should be annotated with the following content in the relationField: *field=IdPerson;table=TablPersonAddress;field=IdAddress;*. The IdPerson is the field that connects the TablPerson to TablPersonAddress. The IdAddress connects the table TablPersonAddress to TablAddress.

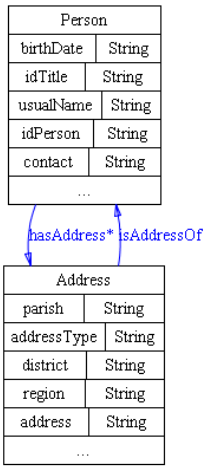

Figure 7: Domain class Contact and Person and Address

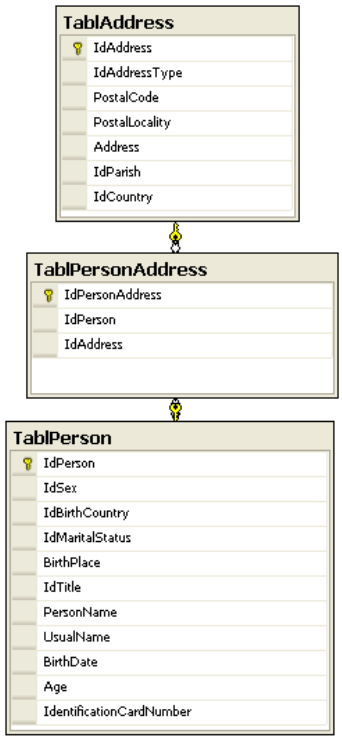

Figure 8: Table Person, Address and the intermediary relation table

Querying the Middleware

As already mentioned and depicted in Figure 3, our middleware is accessible through Web Services and the message content of the request and response is structured in XML and encapsulated in a SOAP envelope. Users interact with our system making requests (which contains 'what data' is needed), using a XML structure named CQL (Client Query Language) defined by the XSD schema illustrated in Figure 9. CQL allow users to specify:

- Fields (ontology attributes) to be returned, using the *outputFields* element.
- The order of the output fields (ascending, descending), using the *orderFields*.
- 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.

These XML elements are described in more detail in the Figure 10 XSD Schema.

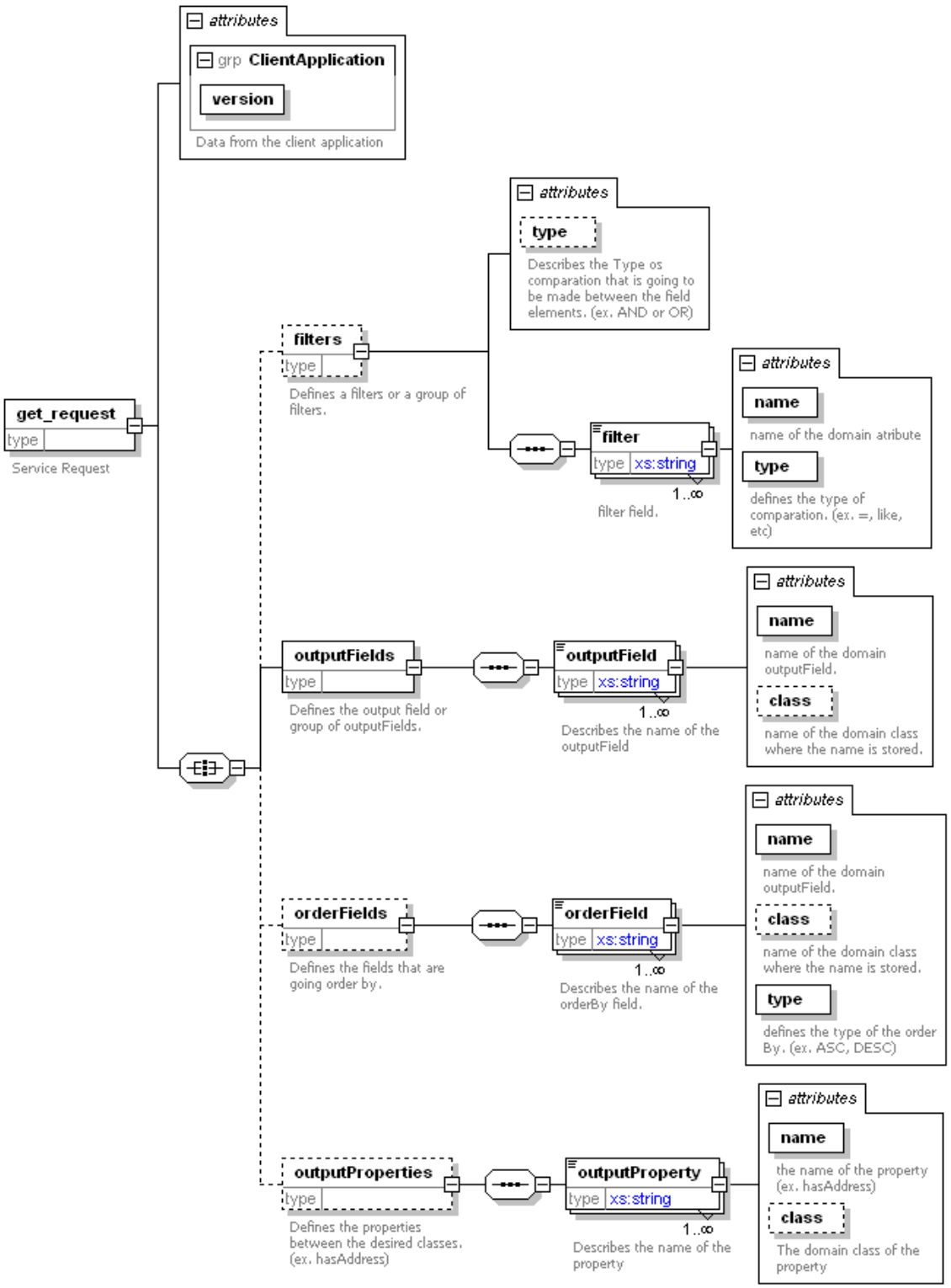

Figure 9: Schema of the XML request (Dashed rectangles represent optional XML elements. All the others are required.)

For example to obtain the birth date and name of a person, the user should make the following request:

```
<get_request>
        <outputFields>
                <outputField name="birthDate"/>
                 <outputField name="usualName"/>
        </outputFields>
        <orderFields>
                 <orderField type="ASC" name="usualName"/>
        </orderFields>
</get_request>
```

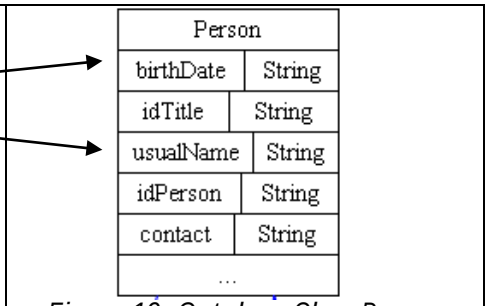

Figure 10: Ontology Class Person

The above-illustrated request is processed by the middleware, therefore generating automatically and on the fly the needed SQL statement to get the data. The needed data in order to build the SQL statement is obtained by executing a SPARQL query (generated by transforming the XML request from the user) on the ontology. The return of the SPARQL query will contain the table name, attribute and relations between tables.
The result of this request (the response) will be the *birthDate* and *usualName* of all 'persons' stored in the database, formatted in XML, sorted by *usualName*. As we will illustrate in the following section, the response of this request is a XML structure that hierarchical describes the ontology.
In the next section, we will illustrate a running example, which shows a small ontology, the database schema that stores the data described by the ontology, the instances that map the ontology concepts to the tables in the database, the request and the response.

Example

Let us suppose that we are interested in getting the name of all our students and their address. The first step is to locate the concepts in the ontology. Next, we need to formulate the request. By invoking the service, we will get the data, structured in XML accordingly to the ontology.

Ontology

In Figure 11, we illustrate the ontology classes Person and Address belonging to the personal data ontology. Some of the fields associated to each class are also illustrated.

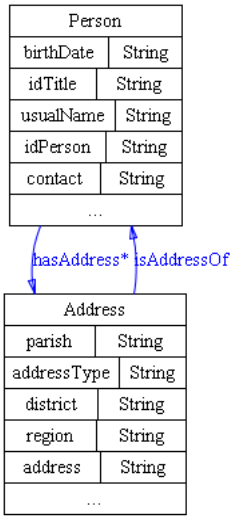

Figure 11: Extract from the complete Person Data ontology

Database Schema

Next, we illustrate an extract from the personal data database schema. In the example, we will focus in the TablPerson, TablPersonAddress e TablAddress.

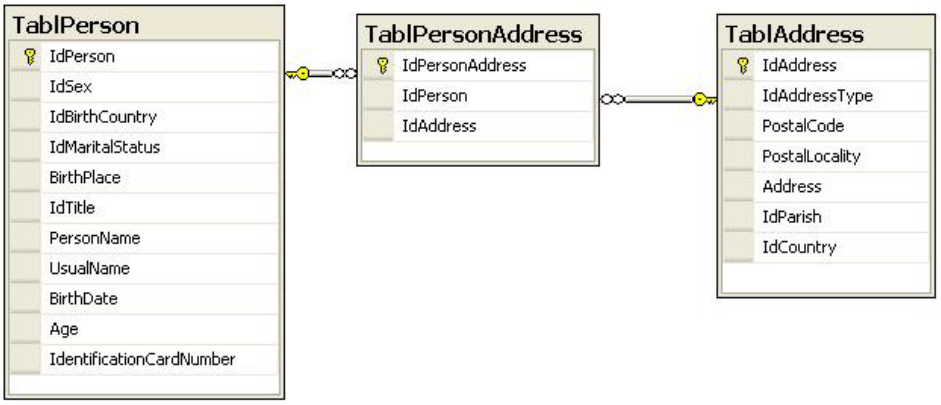

Figure 12: Extract from the Personal Data database

Mappings – Instances

In this section, we illustrate the mapping instances of the domain classes Address and Person, highlighting the TablPerson and its attribute birthDate and the TablAddress and its attribute address. The hasAddress object property is annotated with the following content in the relationField: field=IdPerson;table=BDMest.dbo.TablPersonAddress;field=IdAddress;. This annotation relates the two tables through the table TablPersonAddress. The attributes that relate TablPerson with TablPersonAddress (which is IdPerson) and TabkPersonAddress with TablAddress (which is IdAddress) are also stored in the annotation.

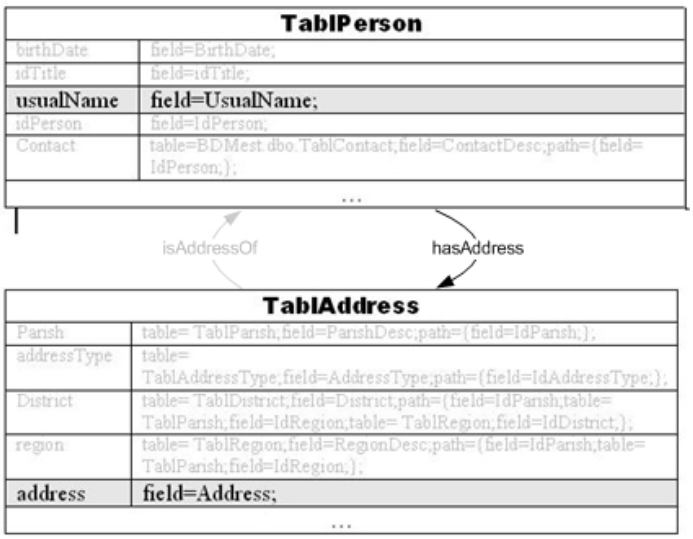

Figure 13: Instances of the classes of the ontology

Request

We are interested in getting the name and the address information of a person. The correct query in order to get the right information is:

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

The hasAddress object property connects the two required classes Address and Person.
The middleware generates automatically and on the fly the SQL statement by extracting the 'mapping information' stored in the instances of the ontology.

```
SELECT
      TablPerson.PersonName AS name,
      TablAddress.Address AS address
FROM TablPerson
LEFT JOIN TablPersonAddress
 ON TablPerson.IdPerson = TablPersonAddress.IdPerson
LEFT JOIN TablAddress
 ON TablPersonAddress.IdAddress = TablAddress.IdAddress
```

Response

The returned data, formatted in XML accordingly to the domain layer, would be:

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

The structure of the XML is accordingly to the ontology structure. The main reason that motivated this approach was that, in the current application scenario, it is easier to clients (php, java …) to read and manipulate raw XML than an OWL structure.

Related Work

Several tools and approaches to integrate heterogeneous data sources exist today. We will briefly describe the Corporate Ontology Grid (COG), the Mediator envirOnment for Multiple Information Systems (MOMIS), OBSERVER**,** the Knowledge Reuse And Fusion/Transformation (KRAFT) and InfoSleuth.

Some of the approches are based on agents. Examples are InfoSleuth and KRAFT. **InfoSleuth** is a multi-agent system for semantic interoperability in heterogeneous data sources [NFKPTU99]. Agents are used to query and instance transformations between data schemas. An agent is aware of its own ontology and the mapping between that 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, made available through the ontology agents.
KRAFT 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 [GPFGB97].
In both systems, a language implementing the protocol of communication agents and a language expressing the information to be extracted is needed.

OBSERVER is a component-based approach to ontology mapping. 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. It 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 [MKSI96]. Query language in the OBERVER is specific. In each node, component mapping must exist to all other relevant nodes (one-to-one mapping).

The **COG** aims to create a semantic information management in which several heterogeneous data sources are integrated into a global virtual view [Bru04]. 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 accommodates both the GAV and LAV approach [Bru04, BJ04]. Queries cannot be executed in the workbench and it is not possible to query multiple data sources. Views over the global virtual view have to be previously created in order to permit the access to the data. Queries are very similar to SQL. The access to the global virtual view is done via specific API. Compared to COG, our approach only allows integration of RDBMS data sources. Queries are generated dynamically and it is possible to execute queries over different databases. Requests are formatted in XML. Our approach allows the provision of services through Web Servives.

The goal of **MOMIS** is to give the user a global virtual view of the information coming from heterogeneous data sources [BB04, BBCG04]. 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. It is based in GAV approach, which means that the global schema is built based on the local sources. OWL and SPARQL are not used in MOMIS. Queries are expressed using a SQL-like language.
None of the solutions presented use OWL as a way of describing the domain model and storing the mappings to the databases. Much of the presented solutions are somehow limited in generating the statement to get the data. In others, a specific query language was created using proprietary language.

Conclusion

In this paper, we have presented our solution for creating a middleware to provide integration and abstraction between clients and databases. Our system creates a global virtual view over a set of data sources, using ontologies specified in OWL. We used a layered system, allowing reuse, evolution and incremental development. Therefore, three layers compose the middleware: data source, domain and interface. The interface layer provides services and allows the interoperability of our solution with other systems/organizations, structuring requests and responses in XML, exposed as Web Services. A customize request language, expressed in XML, allows users to interact with the system, abstracting from technical details.
The domain layer uses OWL ontologies, describing the domain and allowing the integration of several data sources. Mappings to the data sources are stored in the instances of the ontology. With this solution, we can distribute our ontology among stakeholders because it does not contain neither confidential nor technical data.
The data source layer is implemented using Hibernate, which allows connection to more that 20 database vendors using JDBC.
We are convicted that our solution will guarantee and improve the integration of heterogeneous data sources by the use of a semantic abstraction layer, described in OWL. It also decouples client applications from database servers, minimizing maintenance.

References

[ABBFLL05] Alexiev, V.; Breu, M.; Bruijn, Jd.; Fensel, D.; Lara, R.; Lausen, H.: Information Integration with Ontologies, *John Wiley &Sons*, 2005

[AH04] Antoniou, G.;, Harmelen, F.: A Semantic Web Primer, *MIT Press*, 2004

[Bru04] Bruijn, Jd.: Semantic Integration of Disparate Data Sources in the COG Project, www.debruijn.net/publications/COG-ICEIS2004.pdf , 2004

[BB04] Beneventano, D.; Bergamaschi, S.: The MOMIS Methodology for Integrating Heterogeneous Data Sources, www.dbgroup.unimo.it/prototipo/paper/ifip2004.pdf , 2004

[BBCG04] Bergamaschi. S.; Beneventano, D.; Corni, A.; Gelati, G. and others: The MOMIS System, http://www.dbgroup.unimo.it/Momis/ ,2004

[BJ04] Bruijn, Jd.: Best Practices in Semantic Information Integration, 2004

[BM02] Berners-Lee, T.; Miller, E.: The Semantic Web Lifts Off, *Special Issue of ERCIM News*, 2002

[CA06] Cardoso, J.; Sheth, A.: Semantic Web Services, Processes and Applications, *Springer*, 2006

[Dar97] Darleen, S.: Client/Server Software Architectures – An Overview, http://www.sei.cmu.edu/str/descriptions/clientserver.html, 1997

[DS00] Darleen, S.; Santiago, C: Three tier software architectures, http://www.sei.cmu.edu/str/descriptions/threetier.html, 2000

[FC] http://www.webopedia.com/TERM/F/fat_client.html

[FRFHM02] Fowler, M.; Rice, D.; Foemmel, M.; Hieatt, E.; Mee. R.; Stafford, R.: Patterns of Enterprise Application Architecture, *Addison-Wesley*, 2002

[FWK02] Fremantle, P.; Weerawarana, S.; Khalaf, R.: Enterprise Services, *Communications of the ACM*, 2002

[GPFGB97] Grayy, P.; Preecey, A.; Fiddianz, N.; Grayz, W.; Bench-Capon T. and others: KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases, www.csd.abdn.ac.uk/~apreece/research/download/dexa1997.pdf , 1997

[Grub93] Gruber, T.: A translation approach to portable ontologies. Knowledge Acquisition, *Academic Press*, 1993

[He03] He, H.: What is Service-Oriented Architecture, http://www.xml.com/pub/a/ws/2003/09/30/soa.html, 2003

[HIB] http://www.hibernate.org/

[HL] http://en.wikipedia.org/wiki/Abstraction_layer

[HW04] Hohpe, G.; Woolf, B.: Enterprise Integration Patterns, Addison-Wesley, 2004

[J06] JBoss: Hibernate Reference Documentation, http://labs.jboss.com/portal/ , 2006

[Lan02] Lanzerini, M.: Data Integration: A Theoretical Perspective, http://www.cs.ubc.ca/~rap/teaching/534a/readings/Lenzerini-pods02.pdf , 2002

[MFK01] Manolescu, I.; Florescu, D.; Kossmann, D.: Answering XML Queries over Heterogeneous Data Sources, www.vldb.org/conf/2001/P241.pdf , 2001

[MKSI96] Mena, E.; Kashyap, V.; Sheth, A.; Illarramendi, A.: OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies, http://dit.unitn.it/~p2p/RelatedWork/Matching/MKSI96.pdf , 1996

[MW] http://en.wikipedia.org/wiki/Middleware

[NFKPTU99] Nodine, M.; Fowler, J.; Ksiezyk, T.; Perry, B.; Taylor. M.; Unruh, A.: Active Information Gathering in InfoSleuth, www.argreenhouse.com/InfoSleuth/publications/codas99.pdf , 1999

[Ogb02] Ogbuji, U.: The Past, Present and Future of Web Services Part 1 and 2, Web Services.org, 2002

[O06] Oracle White Paper, Semantic Data Integration for the Enterprises, *Oracle*, 2006

[Pal01] Palmer, S.: The Semantic Web: An Introduction, http://infomesh.net/2001/swintro/, 2001

[PA03] Polikoff, I.; Allemang, D.: Semantic Integration: Strategies and Tools, *TopQuadrant*, http://www.topquadrant.com/documents/TQ0303_Semantic%20Integration.PDF, 2003

[PS06] Prud'hommeaux, E.; Seaborne, A.: SPARQL query language for RDF, http://www.w3.org/TR/rdf-sparql-query/, 2006
[PTE] http://protege.stanford.edu/
[Rit05] Rito, A.: A Software Architecture for WEB Applications: A Student Management System Example, *Instituto Superior Técnico*, 2005
[Sta02] Stal, M.: Web Services: Beyond Component-Based Computing, *Communications of the ACM*, 2002
[Tay06] Taylor, J.: Enterprise Information Integration: A new Definition, Integration Consortium, http://www.dmreview.com/article_sub.cfm?articleId=1009669, 2006
[TC] http://www.webopedia.com/TERM/T/thin_client.html
[W3CSC] http://www.w3.org/XML/Schema
[W3CSP] http://www.w3.org/TR/soap/
[W3CXS] http://www.w3.org/TR/xslt
[WS] http://www.w3.org/2002/ws/
[Yag02] Yager, T.: The Future of Application Integration, http://www.computerworld.com/action/article.do?command=viewArticleTOC&specialReportId=3&articleId=71198, 2002

Biography

Bruno Caires has received a Post Graduate Diploma in Software Engineering from University of Madeira, Portugal. He is currently pursuing MS in Software Engineering from the same University. Actually, and for the last three years, he is responsible for the development of SOA middleware. His interests include System Integration, SOA, Web Services, Middleware and Semantic Web Technologies.

Jorge Cardoso received his PhD in Computer Science from the University of Georgia in 2002. Actually, he is Professor at University of Madeira. He previously gave lectures at University of Georgia (USA) and Instituto Politécnico de Leiria (Portugal). While at the University of Georgia he was part of the LSDIS Lab, where he did extensive research on workflow management systems. Current interests include Workflow Quality of Service, Semantic Workflow Composition, Web services, Web processes, e-Commerce, and Groupware/CSCW.