USING SEMANTIC WEB AND SERVICE ORIENTED TECHNOLOGIES TO BUILD LOOSELY COUPLED SYSTEMS

SWOAT – A Service and Semantic Web Oriented Architecture Technology

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

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

Keywords: Information Integration, Semantic Web, Ontology, Web Services, Middleware

Abstract: The creation of loosely coupled and flexible applications has been a challenge faced by most organizations. This has been important because organization systems need to quickly respond and adapt to changes that occur in the business environment. In order to address these key issues, we implemented SWOAT, a 'Service and Semantic Web Oriented Architecture Technology' based middleware. Our system uses ontologies to semantically describe and formalize the information model of the organization, providing a global and integrated view over a set of database systems. It also allows interoperability with several systems using Web Services. Using ontologies and Web services, clients remain loosely coupled from data sources. As a result, data structures can be changed and moved without having to change all clients, internal or external to the organization.

1 INTRODUCTION

A middleware for data integration should allow users to focus on 'what' information is required and leave the details on 'how' to obtain and integrate information hidden from users. Thus, in general, data integration must provide mechanisms to communicate with data sources, handle queries across heterogeneous data sources and combine the results in an interoperable format, returning it to the clients (Silva and Cardoso, 2006). During the software lifetime, several changes that occur are caused by maintenance tasks, either corrective, adaptative, preventive or perfective (Vliet, 2000). Client applications, when directly connected to the database system, remain vulnerable to database changes, most of them motivated by maintenance tasks. To address these issues, we needed a solution that takes all the advantages from the data integration middleware, decoupling client applications from database systems. We believe that in order to achieve the above-mentioned objectives, the solution resides in developing middleware with two emerging technologies: *Semantic Web Technologies* (SWT) and *Service Oriented Architectures* (SOA).

Several reasons motivated the use o **SWT** in our middleware. The four main reasons are (Noy and McGuinness, 2001): (1) To share common understanding of the structure of information among people or software agents. This way, the model can be understood by humans and computers; (2) to enable reuse of already specified domain knowledge. (3) To make domain assumptions explicit; this means that concepts defined in the model have a well defined an unambiguous meaning; (4) Analyze domain knowledge is possible once a declarative specification of the terms is available.

To prove that SWT has already some successful and useful projects implemented, ten application areas have been identified and described (Cardoso and Sheth, 2006). Five examples are: (1) Semantic Web Services; (2) Semantic Integration of Tourism Information Sources; (3) Semantic Digital Libraries; (4) Semantic Enterprise Information Integration; (5) Semantic Web Search.

Ontologies, applied on middleware, in spite of not having been identified as one of the ten potential application areas, can be used in order to provide a global virtual view over a set of database servers (Alexiev et al., 2005). Our implementation has shown that the use of SWT added value to our middleware system, semantically describing (focussing on 'how' and not on 'what') and centralizing the information model of the organization.

The other emerging technology used in SWOAT is **SOA**. It provides a universal access mechanism to all systems via *Web Services* (WS) and a universal data representative via *XML* (Taylor, 2004). Based on *Web* technologies, WS gain several advantages. One of them is that being based on SOAP-over-HTTP (Web based protocols), WS are designed to work over public Internet. Another *Web* aspect is that interoperability is achieved using SOAP that defines a common standard allowing differing systems to interoperate. Also, the XML is a standard framework for creating machine-readable documents (Fremantle et al., 2002). The *Service* (Web + Service) main aspects of WS are that services are available to all systems that wish to use them. Another aspect is that services have a machine-readable description that can be used to identify the interface of the service, its location and access information. Finally, the service interface is available independently of the ultimate system implementation (Fremantle et al., 2002).

Taking advantage of SWT and SOA, we implemented SWOAT in order to use the synergy of these two technologies. Using SWOAT to deploy Information Systems (IS) we have identified three main advantages:

- The request done by clients, formulated to get the required data, specifies *'what information is needed'* and nothing 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 (Cardoso and Sheth, 2006) is solved.

In the next section, we are going to describe the scenario that motivated the development of SWOAT.

2 MOTIVATING SCENARIO

Let us consider an organization with three thousand users, where more than ninety percent of them use the organization main information systems. The organization owns several systems, either developed internally or externally (COTS), and all of them store their data on relational databases. Examples of COTS are the human resource management system and accounting system. The organization core business is developed in-house and because it is the main organization's system, needs to extract and use data stored in several other database systems (like the human resource management system and the accounting system). The core business system has two types of clients: GUI and Web. 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 all authenticated users, from internally or externally of the organization.

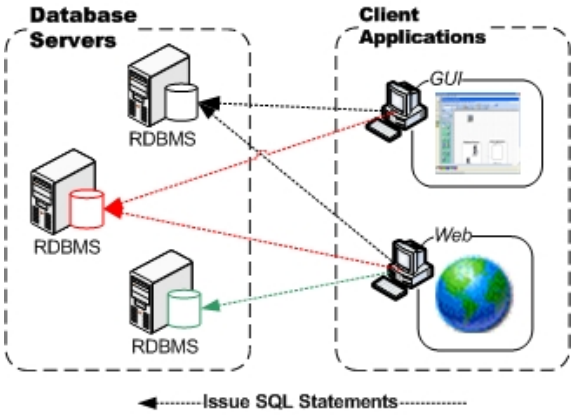

Figure 1: Connections between clients and relational databases

As illustrated in Figure 1, in the current scenario clients are directly connected to databases, which mean that are vulnerable to database changes. For example, changes in a table name may require

changes in several application clients and/or in several database views.

In this scenario, SWOAT will provide a global and integrated view over the set of relational databases and an abstraction layer, decoupling clients (either GUI or Web) from database servers. Client applications either distributed geographically or not can interact with databases mediated by SWOAT, using WS.

3 SWOAT TECHNOLOGIES

SWOAT middleware is based on Semantic Web (Lassila and Swick, 1999), that 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). From the middleware perspective, SW definition contains concepts that are particularly useful in providing a global virtual view over a set of databases: 'well defined meaning' and 'enabling computer and people'. In order to achieve the above-mentioned concepts, **ontologies** are the solution proposed. An ontology is specification of a conceptualization (Gruber, 1993). A conceptualization is the way we think about a specific domain and a specification provides a formal way of writing it down. Ontology represents agreement and common terminology / nomenclature and is in turn the centre price that enables resolution of semantic heterogeneity. Ontology defines a common vocabulary for researchers who need to share information in a domain, including machine-interpretable definitions of basic concepts in the domain and relations among them. Our system represents ontologies using the Web Ontology Language (**OWL**), a semantic mark-up language for publishing and sharing ontologies on the World Wide Web. Other alternative formal languages can also be used to express ontologies, for instance CycL (CyCorp, 2006), KIF (Genesereth, 2006) or RDF (Lassila and Swick, 1999). We choose OWL because it is a W3C recommendation, designed for use by applications that need to process the content of information instead of just presenting information to humans (McGuinness and Harmelen, 2004).

How are ontologies, specified in OWL, used in SWOAT? An ontology is used to provide a unified view over a set of relational database systems, providing a common vocabulary. The ontology is stored in the middleware, therefore centralized and used by all applications that intent to extract data from databases. As is going to be described in the following sections, the mappings from the ontology to the database are stored in the ontology instances.

Several languages that allow to query ontologies exist, namely: RQL, RDQL, N3, SeRQL, SPARQL, Versa, Triple, SquishQL, RxPath and RDFQL (Haase et al., 2004). In SWOAT, we use **SPARQL**, 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). SWOAT uses SPARQL to extract information from the OWL instances in order to generate the SQL statement that allows getting the required data from the database. This is going to be illustrated in more detail in the following sections.

SWOAT allows interoperability through Web services. SWOAT has an interface described in a machine-processable format (specifically **WSDL**, which describes the services provided). Other systems interact with SWOAT using **SOAP** messages (which is intended for exchanging structured information in a decentralized, distributed environment), typically conveyed using **HTTP**.

4 SWOAT ARCHITECTURE

SWOAT is the *middle-tier* that is deployed between the *database tier* and the *client tier*, as illustrated in Figure 2. The database tier contains the databases that store the data and the client tier represents the client applications, internal or external to the organization that invokes the services available by SWOAT, which is the middle-tier.

SWOAT middle-tier is organized in three layers: Data Source Layer (DSL), Business Layer (BL) and Presentation Layer (PL). As illustrated in Figure 2, DSL is identified by (1), BL by (4) and DSL by (11). We are going to describe each layer separately, referring to the numbers illustrated in Figure 2 to facilitate the presentation.

4.1 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,

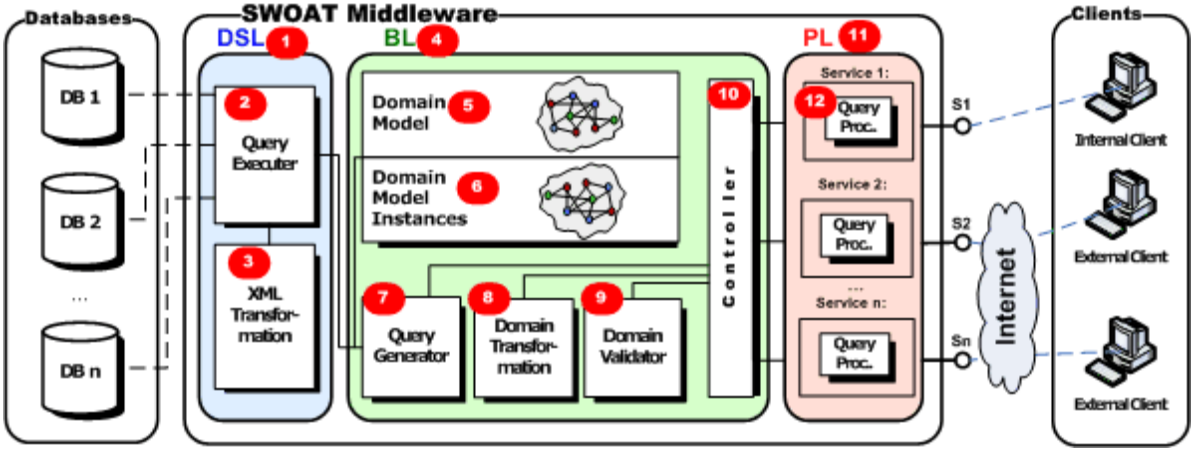

Figure 2: SWOAT Architecture

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 DL, which means that independently from the implementation of the DSL, the only thing that has to be assured is the XML structure.

4.2 Business Layer

The BL (4) is where the domain model, described in OWL is stored. The domain model (5) represents the important concepts, its attributes, and relations between concepts. The concepts are mapped to the DSL, which allows access to the data described by the concept, by creating instances (6) of the OWL model. The instances contain the necessary information (database, table, attribute) in order to build the SQL queries that allows 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 in the domain model is specified that one person has at least one address, the result XML will reflect this hierarchical structure. This is going to be illustrated in the running example section.

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.

4.3 Presentation Layer

The PL (11) is responsible for receiving and processing the requests from the clients. Requests are structured in XML, 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. SWOAT Web Services are implemented using JBoss (JBoss, 2006). JBoss Application Server is one of the most used Java application server on the market, and it is an open source project (JBoss, 2006).

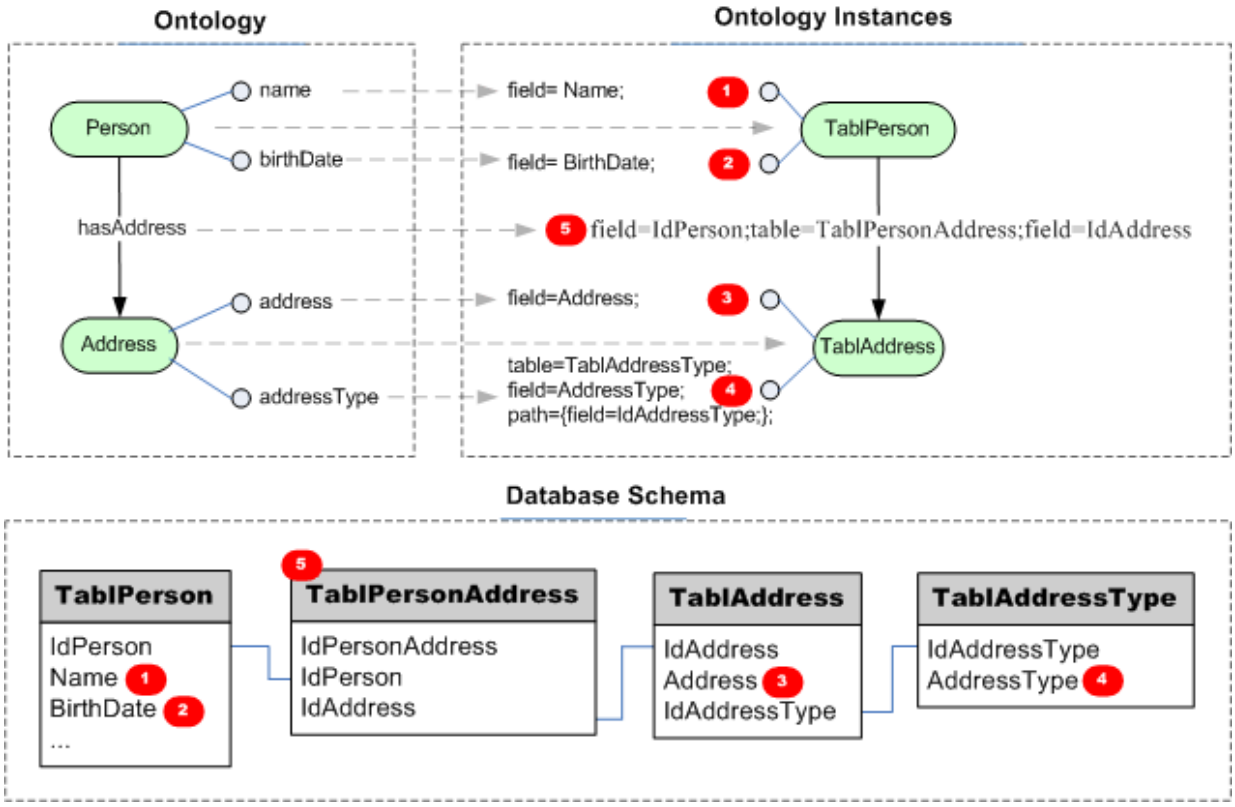

Figure 3: SWOAT Mapings

5 RUNNING EXAMPLE

Let us suppose that we are interested in getting the name of all persons and their address stored in a relational database system. For this example, we will describe the ontology that describes the personal data, namely Person and Address. The database structure that stores the data is also presented. It follows with the ontology instances that contain the mapping to the database layer. XML request and response of the service layer are also illustrated.

In Figure 3 we illustrate the *ontology* classes and their attributes (top left), the *ontology instances* that contain the mappings to the database tables (top right), and the *database schema* that contains the data described by the ontology (bottom).

As depicted in the ontology (top left of Figure 3), for this example, a Person has a name and a birthDate. In the case of the Address, it contains an address description (address) and an addressType. The relation between them is hasAddress, which means that a person *has* an address.

Illustrated at the bottom of Figure 3, the table TablPerson, TablPersonAddress, TablAddress and TablAddressType represent the database structure that store the data described by the ontology. The table TablPersonAddress is used because it is a relation from *n to m* (which means that one person may have one or more addresses and one address may belong to one or more persons).

The ontology instances, illustrated at the top right of Figure 3 contains the mappings from the concepts of the ontology to the database tables. There are instances of the ontology class Person and Address. The name of the instances is equal to the name of the database respective table, as illustrated in Figure 3. For example, TablPerson is an instance of the domain class Person, and its name represents the database table name. The attributes 'name', 'birthDate' and 'address' are stored on the database table that is equal to the ontology instance name, as illustrated by (1), (2) and (3) in Figure 3. Since the 'addressType' attribute is not contained on the TablAddress, but instead on the table *TablAddressType*, a special mapping has to be done: *table=TablAddressType; field=AddressType; path={field=IdAddressType;};* (4). The 'table=' contains the table name that stores

the data, ‘field=’ contains the attribute and ‘path=‘ contains the path from the table TablAddress (name of the instance) to TablAddressType (name of the attrubute in the mapping). As described in the ‘path=’ the attribute that related the two tables is ‘IdAddressType’.

Until now all tables and attributes are mapped, but it is remaining the intermediary table: TablPersonAddress. This type of mapping is addressed in the relation property named hasAddress (5). So, the property hasAddress, that directionally connects two classes, will contain the flowing mapping:
field=IdPerson;table=TablPersonAddress; field=IdAddress;. The field=IdPerson connects the table TablPerson with the table TablPersonAddress. The field IdAddress connects the table TablPersonAddress with TablAddress.

This is all the information that is needed in order to generate the SQL statement with the objective of executing it to return the required data.

5.1 SWOAT Requests and Responses

Requests are structured in XML and allow clients (applications, internal or external, and users) to interact with SWOAT in order to get the desired data. XML requests allow users to specify:

- Fields that should be *returned*, using the XML outputFields element. It is a required element.
- The *order* of the output fields (ascending, descending) is specified using the XML orderFields element.
- *Filters* (for example, return only names started by letter A) are specified using the XML filters element.
- Choose the *path* that connects domain classes, using the *outputProperties* element.

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>
```

This response is structured ‘like’ the domain model, as described in the top left of Figure 3. As already depicted the domain class Person is connected with the class Address through the relation hasAddress.

6 RELATED WORK

Several tools and approaches to integrate heterogeneous data sources and create an abstraction layer exist today (Alexiev et al., 2005). Examples are Corporate Ontology Grid, the Mediator envirOnment for Multiple Information Systems, OBSERVER**,** the Knowledge Reuse And Fusion/Transformation and InfoSleuth.

Some of the approaches, like InfoSleuth and KRAFT, are based on agents. InfloSleuth is a multi-agent system for semantic interoperability in heterogeneous data sources. Agents are used to query and instance transformations between data schemas (Nodine et al., 1999).

In the KRAFT project, users typically have their own local ontology, which is mapped to the central ontology. The basic philosophy of KRAFT is to define a “communication space” within certain communication protocols and languages must be respected (Grayy et al., 1997).

OBSERVER uses multiple pre-existing ontologies to access heterogeneous distributed and independently developed data repositories. It is a component based approach to ontology mapping and provides brokering capabilities across domain ontologies to enhance distributed ontology querying (Mena et al., 1996).

COG aims to create a semantic information management in which several heterogeneous data sources are integrated into a global virtual view (Bruijn, 2004).

MOMIS goal is to give the user a global virtual view of the information coming from heterogeneous data sources (Beneventano and Bergamaschi, 2004).

None of the solutions described are based on Semantic Web technologies defined and specified by W3C. In the described solutions, the ontology, which describes the domain model of the application/organization is described in a non-standard language, most of the times proprietary or adapted in order to address the needed requirements. Interoperability of the described solutions with other applications / organizations is not fully addressed. In fact, XML and Web Services are not used. Getting the data from the data sources involves questioning the middleware. Most of the times, a proprietary "SQL like" query language is used to get the data from the databases, leading to a specific and proprietary query language.

7 CONCLUSION

SWOAT was implemented in order to address the advantages of the three tier architecture. In fact, it acts like an abstraction layer between the client and the database servers. This way, aspects like database location, database technology among others, are transparent to clients. The main objective is that the clients focus on 'what information' and not on 'how to get it' and 'where to get it'. The other objective is to impede that changes on database be propagated to all clients, generating unnecessary maintenance.

Our developed system uses Semantic Web Technologies (SWT), more precisely ontologies, to formally describe the domain model, which is stored and centralized in the middleware. Being a formal model, it is particular suitable to describe and be used by humans and computers.

With the use of service-oriented technology, using Web services, SWOAT allows interoperability with other clients, either internal or external to the organization.

SWOAT is a good solution to quickly create an abstraction layer between clients and database servers, exposing its services as Web Services. Mappings to database are achieved by creating instances of the ontology, allowing that the OWL model can be distributed and reused. Independently of the database structure, the domain model can be mapped to the database tables, exposing information in a format that described the domain model and not the database structure.

To sum up, we can state three main SWOAT characteristics. The first one is that it is an interoperable solution through Web Services (open standards). The second is that it uses OWL to describe the domain model, which is a W3C recommendation that semantically describes the domain model. The third one is that SWOAT XML requests allow clients to specify 'what' information they need, in a non-technical way. These three characteristics will allow the construction of loosely coupled systems.

REFERENCES

Alexiev, V. et al., 2005. Information Integration with Ontologies. John Wiley &Sons, Ltd.

Beneventano, D. and Bergamaschi, S., 2004. The MOMIS Methodology for Integrating Heterogeneous Data Sources.

Berners-Lee, T., Hendler, J. and Lassila, O., 2001. The Semantic Web, Scientific America.

Bruijn, J.d., 2004. Semantic Integration of Disparate Sources in the COG Project.

Cardoso, J. and Sheth, A., 2006. Semantic Web Services, Processes and Applications. Springer.

CyCorp, 2006. Cyc KNowledge Base

Fremantle, P., Weerawarana, S. and Khalaf, R., 2002. Enterprise Services. In: C.o.t. ACM (Editor), pp. 77-82.

Genesereth, M., 2006. Knowledge Interchange Format (KIF).

Grayy, P.M.D. et al., 1997. KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases.

Gruber, T., 1993. A Translation Approach to Portable Ontology Specifications. In: http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html (Editor).

Haase, P., Broekstra, J., Eberhart, A. and Volz, R., 2004. A Comparison of RDF Query Languages, Third International Semantic Web Conference.

Hibernate, 2006. Hibernate Reference Documentation.

Horridge, M., Knublauch, H., Rector, A., Stevens, R. and Wroe, C., 2004. A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0.

JBoss, 2006. JBoss Application Server.

Lassila, O. and Swick, R., 1999. Resource Description Framework (RDF) model and syntax specification.

McGuinness, D.L. and Harmelen, F.v., 2004. OWL Web Ontology Language Overview.

Mena, E., Kashyap, V., Sheth, A. and Illarramendi, A., 1996. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies

Nodine, M. et al., 1999. Active Information Gathering in InfoSleuth

Noy, N.F. and McGuinness, D.L., 2001. Ontology Development 101: A Guide to Creating Your First Ontology

Silva, B. and Cardoso, J., 2006. Semantic Data Extraction for B2B Integration, International Workshop on Dynamic Distributed Systems.

SPARQL-W3C, 2005. SPARQL Query Language for RDF - W3C Working Draft 20 February 2006.

Taylor, J., 2004. Enterprise Information Integration: A New Definition. In: I. Consortium (Editor), Thoughts from the Integration Consortium.

Vliet, H.V., 2000. Software Engineering - Principles and Practice. John Wiley & Sons.